UNISYS

A Series COBOL ANSI-74 Test and Debug System (TADS)

Programming Guide

Relative to the Mark 3.7
System Software Release

Priced Item

July 1987
Distribution Code SE
Printed in U S America
1169901

MCP/AS

UNISYS

COBOL ANSI-74 Test and Debug System (TADS)

Programming Guide

Copyright © 1998 Unisys Corporation.
All rights reserved.
Unisys is a registered trademark of Unisys Corporation.

HMP 4.0 and SSR 45.1 June 1998

Printed in USA
Priced Item 8600 2151–002

The names, places, and/or events used in this publication are not intended to correspond to any individual, group, or association existing, living, or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual, living or otherwise, or that of any group or association is purely coincidental and unintentional.

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THIS DOCUMENT. Any product or related information described herein is only furnished pursuant and subject to the terms and conditions of a duly executed agreement to purchase or lease equipment or to license software. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, special, or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

RESTRICTED – Use, reproduction, or disclosure is restricted by DFARS 252.227–7013 and 252.211–7015/FAR 52.227–14 & 52.227-19 for commercial computer software.

Correspondence regarding this publication should be forwarded to Unisys Corporation either by using the Business Reply Mail form at the back of this document or by addressing remarks to Software Product Information, Unisys Corporation, 25725 Jeronimo Road, Mission Viejo, CA 92691–2792 U.S.A.

Comments about documentation can also be sent through e-mail to **doc@unisys.com**.

Unisys and ClearPath are registered trademarks of Unisys Corporation.
All other terms mentioned in this document that are known to be trademarks or service marks have been appropriately capitalized. Unisys Corporation cannot attest to the accuracy of this information. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

UNISYS

Product Information Announcement

o New Release ● Revision o Update o New Mail Code

Title

MCP/AS COBOL ANSI-74 Test and Debug System (TADS) Programming Guide (8600 2151–002)

This announces a retitling and reissue of the *ClearPath HMP NX and A Series COBOL ANSI-74 Test and Debug System (TADS) Programming Guide*. No new technical changes have been introduced since the HMP 1.0 and SSR 43.2, release in June 1996.

To order a Product Information Library CD-ROM or paper copies of this document

- United States customers, call Unisys Direct at 1-800-448-1424.
- Customers outside the United States, contact your Unisys sales office.
- Unisys personnel, order through the electronic Book Store at http://iwww.bookstore.unisys.com.

Comments about documentation can be sent through e-mail to **doc@unisys.com**.

Announcement only:	Announcement and attachments: AS254	System:	MCP/AS
		Release:	HMP 4.0 and SSR 45.1
		Date:	June 1998
		Part number:	8600 2151–002

Contents

About This Guide ix

Section 1. General Command Functions

Interactive Help Text 1–1
Displaying and Setting Program Data 1–1
Breakpoints 1–2
Tracing Program Flow 1–3
Interactive Performing of Paragraphs or Sections 1–3
Specifying and Maintaining Conditions 1–3
Test Coverage and Frequency Analysis 1–4
Obtaining Commands from a Disk File 1–5
Routing Diagnostic Information 1–5
Terminating a Test Session 1–5

Section 2. Preparing for a Test Session

Preparing Software for Test 2–1
Getting TADS Messages in Other Languages 2–2

Section 3. Using TADS

Invoking the Test System 3–1
Interactive Test and Debug Commands 3–2
Batch Mode TADS 3–2
Ending the Test Session 3–4
Test and Debug Process Checklist 3–4

Section 4. Complex Testing Environments

MARC and COMS Environments 4–1
Testing Libraries 4–2
Multitasking 4–3

Section 5. TADS Commands

AT 5–1
BREAK 5–1
BYE 5–2
CLEAR 5–3
Conditional-Commands 5–4
CONDITIONS 5–8
CONTINUE 5–9
COVERAGE 5–9
DISABLE 5–10
DISCARD 5–11
DISPLAY 5–12
DO 5–16
ENABLE 5–17
END 5–18
EXIT 5–18
FREQUENCY 5–18
HELP 5–20
HISTORY 5–21
MERGESTAT 5–22
ON 5–23
OPTION 5–23
OUTPUT 5–24
PERFORM 5–25
SAVESTAT 5–27
SET 5–28
STEP 5–31
TEACH 5–32
TRACE 5–33
WHEN 5–34

Section 6. Example TADS Session

Appendix A. Command Summary

Appendix B. COBOL74 TADS Compare and Set Rules

Appendix C. Accessing DMSII and TPS Special Constructs

DMSII DMSTATUS C–1
TPS Compile-Time Functions C–1
TPS Record Control Items C–2

Appendix D. Supported Data Types

Appendix E. General Format Notation

Appendix F. Basic Constructs

Index 1

Contents

Tables

3–1. Test and Debug Process Checklist 3–4

A–1. Summary of TADS Commands A–1

B–1. COBOL74 TADS Compare Rules B–1
B–2. COBOL74 TADS Set Rules B–5

E–1. General Format Notation Components E–2

Tables

About This Guide

Purpose

This guide describes the COBOL74 Test and Debug System (TADS), which is an interactive tool for testing and debugging COBOL74 programs and COBOL74 libraries. TADS enables the programmer to monitor and control the execution of the software under test and examine the data at any given point during program execution. Interaction with TADS is in terms familiar to programmers, such as paragraph names, variable names, and source statement sequence numbers.

TADS messages can be displayed in natural languages other than English (such as Spanish and German). This feature is made possible by the Message Translation Utility and the MultiLingual System (MLS).

Scope

This guide explains the basic steps for using TADS, including preparing a program for testing, creating a test environment, initiating a test session, and using the system's interactive commands. Other methods that are also explained include testing libraries and using TADS in the multitasking, Menu-Assisted Resource Control (MARC), and Communications Management Systems (COMS) environments. Detailed descriptions and examples of TADS commands are provided. Information about TADS commands can also be accessed during a test session by using the interactive HELP command. Information about the TADS commands is displayed at your station when you use the HELP or TEACH command.

Audience

This guide is for COBOL74 programmers who want to use TADS to test and debug their programs.

Prerequisites

This tool does not require that you be familiar with the details of the systems hardware architecture.

How to Use This Guide

Sections 1 through 5 of this guide provide an introduction to using TADS and describe the general functions of the TADS commands. These sections also include procedures for preparing the software to be tested, invoking TADS, using interactive TADS commands, and concluding the test session. Using TADS on libraries and in multitasking environments is also discussed. Because using TADS on libraries and in multitasking environments is a complex task, you can skip Section 4, "Complex Testing Environments," until you need to use TADS for these purposes.

Section 6 and the appendixes provide reference information that enables you to select and review the TADS commands. The commands are described in detail, and examples of their use are given. The appendixes summarize the TADS commands, describe the COBOL74 TADS compare and set rules, and explain how to read COBOL syntax diagrams that are used to graphically represent the syntax of TADS commands.

If you are unfamiliar with TADS, you might benefit by first reading Sections 1 through 3 ("General Command Functions," "Preparing for a Test Session," and "Using TADS," respectively).

If you are already familiar with TADS, you can use Section 5, "TADS Commands," as a reference for the syntax and function of particular commands. This section uses COBOL syntax diagrams to show proper syntax for each command. Metatokens used in the diagrams are defined for the command; however, if you would like additional information about the metatokens, refer to Appendix F, "Basic Constructs." A summary of the commands is available in Appendix A, "Command Summary."

Organization

This guide consists of the following sections and appendixes. In addition, an index appears at the end of this guide.

Section 1. General Command Functions

The TADS commands are grouped and described according to their general function.

Section 2. Preparing for a Test Session

The steps involved in making TADS available, preparing the test program, and creating the testing environment are described.

Section 3. Using TADS

Invoking TADS, using the interactive TADS commands, and terminating the test session are described. A test and debug process checklist is also provided.

Section 4. Complex Testing Environments

Testing libraries and using TADS in the multitasking, MARC, and COMS environments are explained.

Section 5. TADS Commands

The TADS commands are alphabetically listed and syntactically defined.

Section 6. Example TADS Session

An example of using TADS commands in a test session is provided.

Appendix A. Command Summary

The TADS commands are listed with abbreviations and short descriptions.

Appendix B. COBOL74 TADS Compare and Set Rules

Charts describing COBOL74 TADS compare and set rules for reference are provided.

Appendix C. Accessing DMSII and TPS Special Constructs

Some Data Management System II (DMSII) and transaction processing system (TPS) constructs without COBOL74 TADS syntax are shown with their TADS equivalent constructs.

Appendix D. Supported Data Types

The COBOL74 receiving items that are supported by TADS are listed.

Appendix E. General Format Notation

Instructions on how to read formats are provided.

Appendix F. Basic Constructs

A description of the basic constructs is provided.

Related Product Information

Unless otherwise stated, all documents referred to in this publication are MCP/AS documents. The titles have been shortened for increased usability and ease of reading.

The following documents are included with the software release documentation and provide general reference information:

- The *Glossary* includes definitions of terms used in this document.
- The *Documentation Road Map* is a pictorial representation of the Product Information (PI) library. You follow paths through the road map based on tasks you want to perform. The paths lead to the documents you need for those tasks. The Road Map is available on the PI Library CD-ROM. If you know what you want to do, but don't know where to find the information, start with the *Documentation Road Map*.
- The *Information Availability List* (IAL) lists all user documents, online help, and HTML files in the library. The list is sorted by title and by part number.

The following documents provide information that is directly related to the primary subject of this publication.

COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation

This manual describes the basic features of the standard COBOL ANSI-74 programming language, which is fully compatible with the American National Standard, X3.23-1974. This manual is written for programmers who are familiar with programming concepts.

COBOL ANSI-74 Programming Reference Manual, Volume 2: Product Interfaces

This manual describes the extensions to the standard COBOL ANSI-74 language. These extensions are designed to allow application programs to interface with the Advanced Data Dictionary System (ADDS), the Communications Management System (COMS), the Data Management System II (DMSII), the DMSII transaction processing system (TPS), the Screen Design Facility (SDF), the Screen Design Facility Plus (SDF Plus), and Semantic Information Manager (SIM) products. This manual is written for programmers who are familiar with COBOL74 programming language concepts and terms.

Message Translation Utility (MSGTRANS) Operations Guide

This guide describes how to use the Message Translation Utility (MSGTRANS) to translate compiled program messages from any natural language to any other natural language. It provides complete instructions for running and using the screen and batch interfaces of MSGTRANS. This guide is written for programmers and translators who create and translate program messages in a MultiLingual System (MLS) environment.

Section 1
General Command Functions

This section provides an overview of TADS.

Interactive Help Text

The file BOOK/COBOL74TADS/ENGLISH must be on your system for you to use the interactive help text.

The commands used to invoke the help text are

- HELP
- TEACH

Entering HELP or TEACH at your terminal or your remote station causes a list of commands to be displayed. These commands can also be used to display a description of a specific command, part of a command, or a metatoken at your station.

Displaying and Setting Program Data

The commands used to display and change program data are

- DISPLAY
- SET

The DISPLAY and SET commands allow values associated with declared elements of the test program to be displayed or changed. The commands can reference individual items, ranges of items, or all the items of a COBOL program.

The DISPLAY command can be used to evaluate expressions, interrogate attributes, and identify the types of program variables and the locations of their declarations. In addition, the DISPLAY command can be used to cause a complete symbolic program dump that displays all active items and their values.

The SET command can be used to change the values of variables or attributes.

Breakpoints

The commands associated with breakpoints are

- BREAK
- CONTINUE
- STEP

Breakpoints are interruptions that allow TADS commands to be entered while the execution of the test program is suspended. The commands that deal with breakpoints are the BREAK command, which can be used to create breakpoints; the CONTINUE command, which continues execution of the test program from a breakpoint; and the STEP command, which continues execution through a specified number of test program statements before breaking.

Whenever a breakpoint occurs, the position of the break and the reason for the breakpoint are displayed. The MIXNUMBER and JOBNUMBER attributes of the current task are displayed if they are different from those at the initial breakpoint.

There are six types of breakpoints.

- The beginning of a test session

  A breakpoint occurs automatically before the first executable statement of a program or library.

- The BREAK command

  The BREAK command is used in conjunction with a conditional-command to cause breakpoints to occur under one of the following:

  - At specific statements in the test program
  - When a conditional-expression specified in the conditional-command is equal to TRUE

- The STEP command

  The STEP command continues execution of the test program and causes a breakpoint after executing a specified number of statements.

- CANDE ?BREAKPOINT (?BP) command

  At any time during a test session, the CANDE command ?BREAKPOINT (?BP) can be entered to suspend the test program. The breakpoint occurs at the beginning of the next test program statement. If the command is entered while execution is in code external to the test program, the breakpoint occurs when execution returns to the test program code. (For additional information on ?BREAKPOINT, refer to the *CANDE Operations Reference Manual.*)

- Program fault

  Program faults that are not handled by the test program are caught by TADS. All fatal program conditions are detected except for stack overflow. Programmatic discontinuation (P-DS) is also caught, but not operator discontinuation (O-DS). Commands can then be entered to interrogate program data, but execution cannot be

continued. However, if the condition occurs in a paragraph or section invoked with the PERFORM command, the EXIT command can be used to return to the breakpoint from which the PERFORM operation was done; execution can then be continued.

When a library is being tested, errors in programs calling the library are not detected by TADS, even when the faults occur in the library's procedures. However, faults are detected if the PERFORM command is used to invoke library procedures. (Refer to the information about complex testing environments for information on testing libraries.)

- TADS error condition

  If an error condition occurs during the evaluation of a conditional-command, a message is displayed at your remote station. You must then indicate if execution is to continue.

Tracing Program Flow

The commands used to trace program flow are

- HISTORY
- TRACE

The HISTORY and TRACE commands can be used to display the execution flow of a program. The HISTORY command gives the history of active paragraphs or sections, and the TRACE command identifies the current position of the program.

Interactive Performing of Paragraphs or Sections

The commands used for performing and exiting paragraphs or sections are

- PERFORM
- EXIT

At any breakpoint, the PERFORM command can be used to invoke any paragraph or section in the test program. At breakpoints during execution of the paragraph or section, TADS commands, including the PERFORM command, can be used. Control is then returned to your station at the breakpoint where the PERFORM command was made.

A paragraph or section invoked by the PERFORM command can be prematurely exited by using the EXIT command.

Specifying and Maintaining Conditions

Commands used to establish a condition are

- AT
- WHEN

Commands used to maintain the list of conditions are

- CONDITIONS
- DISCARD

The conditional-commands AT and WHEN specify where and when, respectively, other appropriate TADS commands are to be executed. A command or list of commands can be associated with a condition so that when the condition occurs, the command or list of commands is executed. AT and WHEN allow most of the TADS commands to be invoked when the specified conditions are met.

The TADS commands that cannot be associated with conditions are the PERFORM, CONTINUE, and STEP commands. These commands can be used only at a breakpoint.

Two commands that are particularly useful with conditions are BREAK and TRACE. The BREAK command causes a breakpoint and the TRACE command displays a program's execution flow.

When a condition is established, it is assigned a number by TADS for identification. Also, a label can be associated with a condition in a conditional-command. The label can be either a single identifier or a quoted string of text. The CONDITIONS command can be used to list the established conditions and to see their identification numbers and labels. The DISCARD command discards a condition identified by either its system-generated identification number or its label.

Test Coverage and Frequency Analysis

The commands that provide statement execution statistics are

- CLEAR
- COVERAGE
- FREQUENCY
- MERGESTAT
- SAVESTAT

These coverage and frequency commands display statistics on the execution of specified statements. The CLEAR command initializes the frequency statistics on specified statements to zero. The COVERAGE command lists the statements that were not executed. The FREQUENCY command lists the number of times specified statements were executed. The MERGESTAT command merges saved and current statistics. The SAVESTAT command saves the current statistics information.

Use of these commands requires that the test program be compiled with the FREQUENCY option specified with the TADS compiler control option, that is, $ SET TADS (FREQUENCY).

Obtaining Commands from a Disk File

The DO command obtains commands from a disk file.

The DO command gets a sequence of TADS commands from a disk file rather than the remote file. Such a file can be used for changing variable values, retrieving previously defined conditional-commands, or automating a particular test case.

Routing Diagnostic Information

The OUTPUT command directs the output of TADS commands to a particular device or file.

The OUTPUT command can be used to send session information to a printer, disk file, or your station. Session information is displayed, by default, at your station.

Terminating a Test Session

The commands used to end a test session are

- BYE
- END

The BYE and END commands terminate the test session and execution of the test program. A test session is also terminated upon completion of the test program or when an end of file (EOF) condition is detected for your remote station. An EOF condition can be caused by the CANDE command ?END. (Refer to the *CANDE Operations Reference Manual.*)

Section 2
Preparing for a Test Session

Before testing can begin, the support library function C74TADSSUPPORT must be established to make the TADS software available. The ?SL (System Library) command, a Command and Edit (CANDE) command, can be used to list the available support library functions. If C74TADSSUPPORT is not listed, it must be made available by your computer site manager.

For online help information to be accessible, the support library function HELPSUPPORT must be established. The help file must also reside on the same family and under the same usercode as the C74TADSSUPPORT library to make online help information available. The ?SL command is used to list the support libraries to determine whether HELPSUPPORT is available. If you want the online help information, it must be made available by your computer site manager.

Preparing Software for Test

The program or library to be tested must be compiled with the TADS compiler control option enabled (equal to TRUE). To do this, include the $ SET TADS compiler control record in the program. The TADS compiler control option must appear before the first source statement. However, comments and other compiler control records can appear before the TADS compiler control option.

The two options available with the TADS compiler control option are FREQUENCY and REMOTE.

The FREQUENCY option enables TADS to accept the coverage and frequency commands in the test session. (These commands provide statistics on the execution of specified statements.) The CLEAR, COVERAGE, FREQUENCY, MERGESTAT, and SAVESTAT commands are accepted when the TADS compiler control option is specified as $ SET TADS (FREQUENCY). The FREQUENCY option significantly slows execution of the test program. If coverage and frequency information is not needed, then the FREQUENCY option should not be used.

The REMOTE option enables TADS to share a remote file with the program being tested. Sharing a remote file might be necessary because a program can have only one remote file open for input at a time.

If the REMOTE option is used, a file title must be specified corresponding with a file that has been selected and assigned to REMOTE. The record associated with the file must be declared with a record containing at least 72 characters.

When the TADS REMOTE option is specified by $ SET TADS (REMOTE FYLE), for example, the program's remote file FYLE is shared with TADS for the test session interaction.

The FREQUENCY and REMOTE options can be used together by specifying, for example, $ SET TADS (FREQUENCY, REMOTE FYLE). In this case, the program's remote file FYLE is shared for test session interaction.

(For additional information on the TADS compiler control options FREQUENCY and REMOTE, refer to the *COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation.*)

The COBOL74 program to be tested can invoke libraries or intrinsics that are compiled with the TADS option equal to FALSE or are written in other languages. However, the test program cannot have been bound and cannot be the result of a COBOL SEPCOMP compilation. Refer to the *COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation* for details of the SEPCOMP compiler control option.

Programs being tested usually do not stand alone. They often require various items external to them, such as files and databases. These external items make up the external testing environment for the program. One way to create a testing environment is to place copies of the desired items under a usercode reserved for testing. Running the tests with copies of the items prevents unwanted changes to the originals.

Getting TADS Messages in Other Languages

TADS messages can be displayed in natural languages other than English (such as German and Spanish). This requires the TADS messages that reside in an output message array to be translated to the desired language or languages by using the Message Translation Utility. TADS can then interface with the MultiLingual System (MLS) to get messages in a specified language.

The process and procedure for translating program messages is described in the *Message Translation Utility (MSGTRANS) Operations Guide.* Refer to the *MultiLingual System (MLS) Administration, Operations, and Programming Guide* for information about accessing the translated program messages.

Section 3
Using TADS

This section describes the ways to run TADS.

Invoking the Test System

TADS is invoked by executing the program to be tested with the TADS task attribute assigned the value TRUE. If a program titled XYZ is compiled with the TADS compiler control option equal to TRUE, the following Command and Edit (CANDE) command initiates a test session:

```
EXECUTE XYZ; TADS
```

The following WFL job initiates a TADS session for a batch program:

```
BEGIN JOB C74TADS;
  EXECUTE OBJECT/XYZ;TADS;
  END JOB
```

The following WFL job initiates a TADS session with an online program:

```
BEGIN JOB C74TADS;
  TASK T;
  T(STATION=MYSELF(SOURCESTATION));
  EXECUTE OBJECT/XYZ[T];TADS;
  END JOB;
```

The same program, compiled with the same options but executed without the TADS task attribute equal to TRUE, runs normally without activating TADS. The TADS task attribute is checked at program initialization to determine if TADS is to be invoked. The value of the attribute cannot be changed after the program is active. More information about task attributes can be found in the *Task Attributes Programming Reference Manual.*

When the test session is initiated, the test program is suspended before the first statement of the program is executed. At this point, commands to TADS can be entered.

Interactive Test and Debug Commands

Interaction with TADS is done through the remote station from which the program was executed. Commands can be entered at the remote station and can also be invoked from a disk file by using the DO command.

The following rules apply to TADS commands:

- A percent sign (%) that is not part of a quoted literal marks the remainder of the line as a comment.
- Uppercase and lowercase letters are treated as equivalent except in quoted literals.
- Multiple commands can be entered on one line if separated by semicolons (;). Also, from your remote station, the carriage return character can be used to separate commands that are entered on one transmission.
- The end of a transmission from your station marks the end of a command. However, the end of a record in a disk file invoked with a DO command does not mark the end of a command.

Note: *All commands in disk files must be terminated by semicolons (;). An end of file (EOF) implies a semicolon, which marks the end of a command.*

The commands described in this guide are graphically represented using COBOL syntax diagrams. Abbreviations that are longer than the minimum are acceptable. For example, C is the minimum abbreviation for the CONTINUE command, but CO, CON, CONT, and so on are all acceptable. Refer to Appendix E, "General Format Notation," for help on COBOL syntax diagrams.

Commands can be entered at any breakpoint. A breakpoint occurs automatically at the beginning of a test session. Breakpoints can be established with the BREAK and STEP commands. Also, the CANDE command ?BREAKPOINT, abbreviated ?BP, can be entered to immediately suspend the program at any time during the session. In addition, when a program fault occurs that is not handled by the program, execution is suspended so that commands can be entered to interrogate program data.

Batch Mode TADS

Batch mode TADS enables a series of identical commands to be entered over several program runs. Therefore, it is not necessary to reenter identical commands for each program run. For example, batch mode might be used to gather coverage statistics for a program using a variety of test data.

TADS commands that are to be executed in batch mode must be stored in a special file. The file must be named UIDISKINPUTFILE, and must be accessible. For example, UIDISKINPUTFILE might be accessible under a particular usercode. An example of UIDISKINPUTFILE follows:

Example

```
100 MERGESTAT FROM MERGEFILE1;
200 DISPLAY **\
300 BREAK AT SUB.END;CONT\
400 COVERAGE\
500 SAVES AS MERGEFILE2\
600 BYE\
```

Each set of TADS commands must be terminated by a backward slash (\) character. Commands can be divided among several lines (as shown on sequence lines 100 through 200 in the preceding example).

TADS locks the UIDISKINPUTFILE and reads and executes commands from it. A record of the TADS session is kept in a file named UIDISKOUTPUTFILE, which is created by TADS at run time, and can be examined for the results of each TADS command. Any TADS command can be put in the UIDISKINPUTFILE.

The UIDISKINPUTFILE is used to signal TADS to switch to batch mode. This file must be file-equated to the internal file name TADSBATCHFILE, and must have file attributes KIND=DISK and MYUSE=IN. These attributes are used by TADS to determine whether or not to switch to batch mode. Note that the UIDISKINPUTFILE must be file-equated to the software that is being tested.

The following example shows how to set up a job deck to run batch mode on regular programs, that is, programs that are not compiler or library programs.

Example

```
BEGIN JOB BATCHTEST;
      RUN OBJECT/COV/TEST;TADS;
      FILE TADSBATCHFILE (KIND=DISK,MYUSE=IN,
      TITLE=UIDISKINPUTFILE);
  END JOB
```

The following example is a WFL job that might be used to run batch mode on a compiler.

Example

```
BEGIN JOB TESTTADSCOV;
   TASK
       COMPT          % FOR COMPILES
    ;
      % (JC)RPG37/SPL/COV is the compiler
      COMPILE DISK/1 WITH (JC)RPG37/SPL/COV [COMPT]
         LIBRARY;
         STATION = MYSELF(SOURCESTATION);
         COMPILER TADS;
         COMPILER FILE CARD(TITLE=SOURCE/RPG/CALC/ARITH/101,KIND=DISK);
         COMPILER FILE TADSBATCHFILE (KIND=DISK,
                                      MYUSE=IN,
                                      TITLE=UIDISKINPUTFILE);
```

```
        DISPLAY("EXIT, COMPT= " & STRING(COMPT(VALUE),*));
END JOB.
```

Ending the Test Session

A test session ends when the program terminates execution or when you end it with the BYE or END command.

Test and Debug Process Checklist

To ensure a successful program test session, consider the issues presented in Table 3–1.

Table 3–1. Test and Debug Process Checklist

Issue	Check	Make available by...
Is the support library function C74TADSSUPPORT available?	The CANDE command ?SL (Support Library)	Checking with your computer site manager to verify that the installed system software is capable of TADS support. That software includes a language compiler, CANDE, the master control program (MCP), and C74TADSSUPPORT library.
Is the compiler control record included in the test program?	$ SET TADS in the test program	Inserting $ SET TADS in the test program. If test coverage and frequency analysis statistics are desired, include the FREQUENCY option. If a remote file is to be shared, include the REMOTE option.
Is the test environment created?		Duplicating any items external to the program, such as files and databases, so as not to disturb the original data.
Are you ready to use TADS in the test session?	TADS task attribute assigned the value TRUE	Entering the CANDE command EXECUTE <file title>; TADS, or starting a WFL job containing the statement EXECUTE <file title>; TADS.
Have you decided to terminate a test session?		Waiting for normal end of task (EOT) or entering the BYE or END command.

Section 4
Complex Testing Environments

This section describes the MARC and COMS environments, libraries, and multitasking.

MARC and COMS Environments

Preparing a program for test in the Menu-Assisted Resource Control (MARC) or Communications Management System (COMS)environment requires some additional preparation. The program must contain a file assigned to REMOTE, which is declared with a record containing at least 72 characters. This file can only be opened I-O (input-output) because it is shared with TADS for TADS session input and output. If a file matching this description does not exist in your program, you need to add one. The program does not have to open the file.

At compile time, the TADS option must be set to TRUE, and the station name at which the TADS session is to be run must be file-equated to the title of the program's remote file. The following COBOL74 program and Work Flow Language (WFL) deck generates a code file, which can be tested by using TADS in the MARC or COMS environment.

COBOL74 Program

```
$SET TADS (REMOTE RMT)
   IDENTIFICATION DIVISION.
   ENVIRONMENT DIVISION.
   I-O SECTION.
   FILE-CONTROL.
        SELECT RMT ASSIGN TO REMOTE.
   DATA DIVISION.
   FILE  SECTION.
   FD RMT.
   01 RMTREC PIC X(80).

   PROCEDURE DIVISION.
   EMPTY SECTION.
   THE-END.
        OPEN I-O RMT.
        STOP RUN.
```

WFL Job

Note: *The station name on which the TADS session is run is TD2611JB.*

```
BEGIN JOB TADS;
  COMPILE OBJECT/XYZ WITH COBOL74 LIBRARY;
        COMPILER FILE CARD (TITLE = XYZ, KIND = DISK);
        FILE RMT (TITLE = TD2611JB);
        TADS;
  END JOB.
```

Running the test program now unconditionally causes a TADS session to start at the station name that is file-equated at compile time to the program's remote file.

At the time the test program is invoked, either by MARC or COMS, the station where the test session interaction is to take place must be logged on.

In the MARC and COMS environments, a remote window is opened for the test session interaction when the program is run. Depending on the number of remote files currently open to the station, the remote window is named REM followed by the window number; for example, REM0001. TADS commands must be entered or passed to the remote window. (Refer to the *Menu-Assisted Resource Control (MARC) Operations Guide* for information on remote windows and the ?PASS command.)

Testing Libraries

The following paragraphs describe how to use TADS to test libraries and how to use TADS in a multitasking environment. Unless you plan to use TADS in these complex environments, you can skip this material.

There are two ways to test libraries:

- A library can be run as a program with the TADS task attribute equal to TRUE. PERFORM commands can be used to execute and test the library's paragraphs or sections.
- A library can be called by other tasks. When the library is first invoked, it behaves like a normal program, taking an automatic breakpoint before its first executable statement. At that point, commands can be entered to cause breakpoints, trace program flow, and so forth. Tasks calling the library stop at breakpoints within the library's procedures.

If a task that calls the library also runs with TADS, then that task must have its own remote station or COMS window for the test session. TADS interaction with the library is done at the library's remote station and TADS interaction with the task occurs at the task's remote station. A library can be associated with a remote station by compiling the library with a value assigned to the STATION task attribute for the desired remote station.

To associate a remote station with the library, the library must be compiled with the TADS attribute equal to TRUE and the STATION attribute assigned a value. When the library is then invoked by another task, the test session for the library is associated with the remote station specified by the STATION attribute. The following Command and Edit

(CANDE)command example compiles the library LIB so that when it is invoked, a test session for the library begins at logical station number (LSN) 198.

```
WFL COMPILE OBJECT/LIB WITH COBOL74 LIBRARY;
        COBOL74 FILE CARD = LIB ON DISK;
        TADS; STATION = 198;
```

If a program that calls a library (for example, OBJECT/LIB/CALLER) is started with a Work Flow Language (WFL) statement, it must have the station attribute assigned. The library, in this case, does not need the station attribute assigned at run time. The following job deck is an example of a program begun with a WFL statement that calls the library (from the previous example).

```
BEGIN JOB;
     RUN OBJECT/LIB/CALLER;STATION=198;
  END JOB.
```

Note: *The library (OBJECT/LIB) and its caller (OBJECT/LIB/CALLER) can use the same value for STATION only when COMS is present. This enables assignment of multiple remote files to a single LSN.*

At the time a library is invoked by another task, the station where the test session interaction is to take place must be logged on, but not active. A program (other than the caller) cannot be running from that station, and another remote input file cannot be open to that station. The program or programs that invoke the library cannot be run directly from the station used for the library's test session. Some of these restrictions might be unnecessary, depending on the value of the CANDE network control command option LAISSEZFILE. (Refer to the *CANDE Operations Manual* for information on LAISSEZFILE and REMOTE files.)

If the STATION attribute is assigned a value, an unconditional breakpoint can be caused in the library by specifying the mix number of the library in the CANDE command ?BREAKPOINT. For example, if the mix number of the library is 1456, then ?1456BP causes a breakpoint. Refer to Section 1, "General Command Functions," for more information.

Multitasking

All tasks executing statements in a program being tested are part of the same test session. The conditional-commands, described in Section 5, "TADS Commands," are effective for all tasks.

However, tasks initiated by COBOL74 CALL, PROCESS, or RUN statements execute statements in a program external to the caller. The run-time value of the TADS task attribute is not inherited from the caller. They must be debugged by running them standalone from CANDE or WFL, or by compile-time equation of the TADS and STATION attributes as described for testing libraries.

Section 5
TADS Commands

The TADS commands are described in this section in alphabetical order.

AT

The AT command, a conditional-command, specifies source statements at which TADS commands are to be executed.

BREAK

The BREAK command interrupts the test program.

Format

```
BREAK
```

Syntax Rules

The minimum abbreviation for BREAK is B.

General Rules

The BREAK command interrupts test program execution and gives control to your station. Any of the other TADS commands can then be used.

BREAK is meaningful only when used as part of a conditional-command.

When a break occurs, the following items are displayed at your station:

- Identification of the current program statement
- The name of the paragraph or section being executed
- The system-generated identification number for the condition causing the break
- The label of the condition, if the condition causing the break has a label

Examples

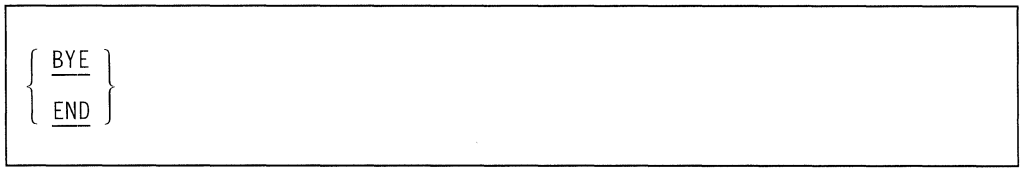
The following examples show the use of the BREAK command.

Example Commands	Results
AT 500 BREAK	Execution stops before the statement at sequence number 500.
BREAK AT 500	Same as AT 500 BREAK.
AT 520.2 BREAK	Execution stops before the second statement at sequence number 520.
AT 600 @ 200 BREAK	Execution stops before execution of the statement at sequence number 200 in the copy library at sequence number 600 in the source program.
WHEN PICXDISPLAY = "COBOL74" AT 600 - 700 BREAK	A breakpoint occurs whenever the program variable PICXDISPLAY equals COBOL74 in the statement range 600 to 700.
WHEN BINARY77SINGLE > 1111 BREAK	A breakpoint occurs whenever program variable BINARY77SINGLE is greater than 1111.
BREAK AT PARA-R ONCE	Execution is suspended only once at the next statement executed in paragraph PARA-R.
BREAK AT SECTION-2.END	A breakpoint always occurs at the end of SECTION-2.

BYE

The BYE command terminates the test session.

Format



General Rules

The BYE command terminates the test session and test program execution. END is a synonym for BYE. The following message signals the end of the TADS session:

```
TADS session ending
```

CLEAR

The CLEAR command clears unsaved frequency statistics for specified statements.

Format

```
CLEAR [ statement-set ]
```

General Rules

The CLEAR command initializes to zero all unsaved frequency and coverage statistics collected during the current TADS session for the specified statements. Further execution of the test program accumulates new statistics.

A statement-set is a set of statements in the test program. If no statement-set is specified, statistics for the entire program are cleared. A detailed definition of statement-set appears in Appendix F, "Basic Constructs."

Examples

Example Commands	Results
CLEAR	Unsaved frequency statistics are cleared for the entire program.
CLEAR 1000.3 - 4000	Unsaved frequency statistics are cleared for the range beginning with the third statement at sequence number 1000 and ending at sequence number 4000.
CLEAR 600 @ 100.2 - 600 @ 300	Unsaved frequency statistics are cleared for the range beginning with the second statement at sequence number 100 in the copy library at sequence number 600 in the source program, and ending at sequence number 300 in the copy library at sequence number 600 in the source program.
CLEAR P1, S-2, 7000	Unsaved frequency statistics are cleared for the statements of paragraph P1 and section S-2 and for the statement at sequence number 7000.

Conditional-Commands

The conditional-commands AT and WHEN specify where and when TADS commands are to be executed.

Format 1

```
{ AT statement-set [ ON expression ] [ ONCE ]                    }
{ ON conditional-expression [ AT statement-set ] [ ONCE ]        }
{ AT statement-set [ WHEN conditional-expression ] [ ONCE ]      }
{ WHEN conditional-expression [ AT statement-set ] [ ONCE ]      }

[ LABEL condition-label ] condition-accepting-command
```

Format 2

```
{ AT statement-set [ ON expression ] [ ONCE ]                    }
{ ON conditional-expression [ AT statement-set] [ ONCE ]         }
{ AT statement-set [ WHEN conditional-expression ] [ ONCE ]      }
{ WHEN conditional-expression [ AT statement-set ] [ ONCE ]      }

[ LABEL condition-label] ({ condition-accepting-command-1 }
                          { conditional-command-1         }

[ ; { condition-accepting-command-2 } ]...)
    { conditional-command-2         }
```

Syntax Rules

The minimum abbreviation for WHEN is W.

If multiple condition-accepting-commands or nested conditional-commands are present, they must be enclosed in parentheses using Format 2. The parentheses and the separating semicolons are required.

In both Format 1 and Format 2 of the conditional-commands, the AT/WHEN, LABEL and condition-accepting-command clauses can be written in any order.

General Rules

The AT, WHEN, and ON commands can be used independently or together to create a condition, although ON and WHEN cannot be used together in the same conditional-command. When the conditions are met, one or more commands are executed. These commands can be condition-accepting-commands and/or other conditional-commands. The parentheses allow the nesting of additional conditional-commands in the original command.

A conditional-command can have two parts:

- AT statement-set
- Either the WHEN conditional-expression or ON statement

One or both of them can be used to specify a condition. The LABEL condition-label is an option for labeling a condition. The ONCE option enables the condition to be discarded after a single execution of the commands associated with it.

Commands that are not allowed as condition-accepting-commands are the PERFORM, CONTINUE, and STEP commands. The CONTINUE command and the STEP command cannot be associated with the conditional-command. The PERFORM command cannot be used as part of the conditional-command because it can be executed only at a breakpoint. A complete list of condition-accepting-commands appears in Appendix F, "Basic Constructs."

A condition has two parts:

- AT statement-set clause
- WHEN conditional-expression clause

One or both of them can be specified.

The AT statement-set clause establishes which test program statements are associated with the conditional-command. The entire program is assumed when AT is not specified.

The statement-set can be a single statement, a range of statements, all the statements in a section or paragraph, or a combination. A detailed definition of statement-set appears in Appendix F, "Basic Constructs."

The conditional-expression of the WHEN clause establishes the conditions under which the command or list of commands is to be executed. The expression is evaluated before each program statement associated with the condition. When the expression is equal to TRUE, the command or list of commands is executed. If the WHEN conditional-expression is not specified, the commands are executed at every specified statement. WHEN TRUE is assumed if the WHEN conditional-expression is not specified.

Conditional-expressions in TADS are the same as those in COBOL74. As in COBOL74 syntax, identifiers can be qualified. Appendix F, "Basic Constructs," discusses the qualification of identifiers and lists the types of constructs that can be used in expressions.

Note that evaluating a conditional-expression contained in the conditional-command over a statement-set of many statements can severely slow program execution.

The ON expression establishes the conditions under which the command or list of commands is to be executed. It is evaluated before each program statement associated with the condition is executed. When the expression has changed from the initialized or previously stored value, the command or list of commands is executed and the current value of the expression is stored for the future comparisons.

Expression qualification and evaluation are the same as described for the WHEN conditional-expression in this section.

The Task, Database, File, Direct File, Library, and Queue variables are not available as expression items for the ON command. The attributes of some of these variables can be used as expressions in the ON command.

The maximum length of data within an array structure that can be specified in an ON expression is 21 words. A truncation warning is given when a length greater than the maximum is requested.

The ON expression and the WHEN conditional-expression each degrade program execution speed by the same degree.

Using the ONCE clause discards the condition at its first occurrence. The command or list of commands associated with the condition is executed only once.

A condition can be labeled with a condition-label: a simple identifier or a quoted string of text. The label is displayed for BREAK and TRACE commands associated with the condition. Multiple conditions can be assigned the same label. Labels can be used to list conditions with the CONDITIONS command and can be used to remove conditions with the DISCARD command.

Examples

Example Commands	Results
AT 100 BREAK	Execution always breaks at the statement having sequence number 100.
AT *.BEGIN, *.END (TRACE)	Trace information is produced at every procedure or block BEGIN and END statement.
BREAK AT P1.END ONCE	Execution breaks the first time it reaches the end of paragraph P1.
AT S2.BEGIN SET X TO 0	Each time section S2 is entered, program variable X is assigned a value of 0.

continued

continued

Example Commands	Results
AT 300, P3 ONCE BREAK	The first time that either the statement at sequence number 300 or any statement in paragraph P3 is executed, a break occurs.
WHEN TRUE TRACE OUT P	Trace information is written to the printer for every statement executed.
WHEN CONDITION-X BREAK	Whenever CONDITION-X is TRUE, a break occurs.
AT 500 WHEN Q > 0 DISP Q	Each time statement 500 is executed, if program variable Q has a value greater than 0, the value is displayed.
AT 500 (WHEN Q > 0 DISP Q)	Each time statement 500 is executed, the conditional-command (WHEN Q > 0 DISP Q) is established.
WHEN DISP03 = "ABC" OR COMP02(I) = @1F@ OR A.[47:3] > B OR CONDITION-NAME-1 AT 600 BREAK	Each time statement 600 is executed, the conditional-expression is evaluated. Each time the expression evaluates to TRUE, the program execution breaks.
AT P4 LABEL IN_P4 (TRACE;DISPLAY Q)	At each statement in paragraph P4, label IN_P4 and the identification of the statement are displayed, followed by information about variable Q.
AT 700 WHEN N = 9 ONCE LABEL "SET BP @ 900"(TRACE; SET N TO 0; AT 900 BREAK)	The first time program variable N has a value of 9 at statement 700, and the label SET BP @ 900 and statement identifier 700 are displayed, then N is assigned a value of 0, and a new condition is established to cause a break at statement 900.
ON P2.X (TRACE)	Each time the value of X in procedure 2 changes, the trace information is produced.
ON (I+6) AT 600 (DISPLAY I+6; BREAK)	Each time statement 600 is executed, the value of (I+6) is evaluated. If it has changed since the last time it was evaluated, the valued of (I+6) is displayed and the program execution breaks.

CONDITIONS

The CONDITIONS command lists established conditions and their corresponding identification numbers.

Format

```
CONDITIONS [ condition-specification ] [ output-command ]
```

Syntax Rules

The minimum abbreviation for CONDITIONS is COND.

General Rules

The CONDITIONS command displays the conditions explicitly specified or all conditions if an asterisk (*) or no list of conditions is specified. An ordinal is the system-generated identification number for a condition. A condition-label is the optional label of a condition. Either the ordinal or the condition-label can be used to identify a condition.

The display of a condition includes its system-generated identification number and the conditional-command that established the condition, including the statement-set, the conditional-expression, the condition-label if there was one, and the command or list of commands associated with the condition.

An attached output-command option directs the output of the list of conditions to a specified device or file.

If no condition-specification is used, all established conditions are displayed. If an asterisk (*) is specified in the condition-specification, all established conditions and conditions disabled with the DISABLE command are displayed.

Examples

Example Commands	Results
CONDITIONS	All conditions are displayed.
COND *	All conditions (established and disabled) are displayed.
COND XYZ, "START OF P"	The conditions labeled XYZ and "START OF P" are displayed.
COND 1, 3 OUT PRINTER	The conditions numbered 1 and 3 are listed on the printer.

CONTINUE

The CONTINUE command causes test program execution to continue.

Format

```
CONTINUE
```

Syntax Rules

The minimum abbreviation for CONTINUE is C.

General Rules

The CONTINUE command causes test program execution to continue from the point where a breakpoint interrupted it. The command can be executed only from a breakpoint; it has no meaning as part of a conditional-command. Also, it cannot be executed at a breakpoint caused by a program fault.

COVERAGE

The COVERAGE command lists the statement-identifiers of statements that were not executed.

Format

```
COVERAGE [ statement-set ] [ output-command ]
```

Syntax Rules

The minimum abbreviation for COVERAGE is COV.

The FREQUENCY option of the TADS compiler control option must be specified for the COVERAGE command to be accepted.

General Rules

The COVERAGE command lists the statement-identifier of each statement in the statement-set that has not been executed. If all statements have been executed, then a message stating that all statements have been executed is displayed. Statement-identifiers are reported either individually or as statement ranges.

If no statement-set is specified, the entire program is considered to be the range. Definitions of statement-set and statement-identifier appear in Appendix F, "Basic Constructs."

An attached output-command option directs the output of the coverage statistics to a specified device or file.

Examples

Example Commands	Results
COVERAGE	A coverage report is created for the program are listed on the printer.
COVERAGE 2000 - 4000	Unexecuted statements in the range 2000 through 4000 are displayed.
COV 600 @ 100 - 600 @ 300.2	Unexecuted statements are displayed in the range from sequence number 100 in the copy library at sequence number 600 to the second statement at sequence number 300 in the copy library at sequence number 600.
COV P-2 OF S-6	Unexecuted statements in paragraph P-2 in section S-6 are displayed.
COV OUT P	All unexecuted statements in the program are listed on the printer.
AT S-2.END COV	When the end of section S-2 is reached, all unexecuted statements in the program are listed.

DISABLE

The DISABLE command causes established conditions to be disabled for later re-establishment with the ENABLE command.

Format

```
DISABLE condition-specification [ output-command ]
```

Syntax Rules

The minimum abbreviation for DISABLE is DISA.

The DISABLE command disables the conditions explicitly specified, or all established conditions if an asterisk (*) is specified.

An ordinal refers to the system-generated identification number for a condition. A condition label refers to the optional label of a condition. Either can be used to disable a specific condition.

General Rules

The results of the DISABLE command are displayed in a message. An error message is given if the specified condition is not available or if no established condition exists.

An attached output-command option directs the output of the generated message to a specific device or file.

Examples

Example Commands	Results
DISA *	All conditions are disabled.
DISABLE 7	Condition number 7 is disabled.
DISA 3, XYZ, "END OF PROC"	Condition number 3, the condition labeled XYZ, and the condition labeled "END OF PROC" are disabled.

DISCARD

The DISCARD command discards established conditions.

Format

```
DISCARD condition-specification [ output-command ]
```

Syntax Rules

The minimum abbreviation for DISCARD is DISC.

General Rules

The DISCARD command discards the conditions explicitly specified, or all conditions if an asterisk (*) is specified.

An ordinal is the system-generated identification number for a condition. A condition-label is the optional label of a condition. Either the ordinal or the condition-label can be used to discard a condition.

An attached output-command option directs the output of the generated message to a specific device or file.

Examples

Example Commands	Results
DISC *	All conditions (established and disabled) are discarded.
DISCARD 7	Condition number 7 is discarded.
DISC 3, XYZ, "END OF PROC"	Condition number 3 and the conditions labeled XYZ and "END OF PROC" are discarded.

DISPLAY

The DISPLAY command displays program elements and values of program variables.

Format 1

```
DISPLAY ** [ data-format-option ] [ output-command ]
```

Format 2

```
                ┌ identifier-1 [ ..identifier-2 ]           ┐
DISPLAY         │ expression-1                               │
                └ ATTRIBUTE attribute-name-1 OF identifier-3 ┘

┌ ┌ identifier-4 [ .. identifier-5 ]           ┐ ┐
│ │ expression-2                               │ │ ...[data-format-option]
└ └ ATTRIBUTE attribute-name-2 OF identifier-6 ┘ ┘

[ output-command ]
```

Syntax Rules

The minimum abbreviation of DISPLAY is D.

Any of the identifiers can be subscripted and/or qualified. The number of subscripts must either match the number required by the item in the record or be null. A double period (..) can be used to represent a range of subscripts. An asterisk (*) represents the entire range, the subscript 1 when used before a double period or the maximum valid subscript when used after a double period.

Qualification must be included if necessary to prevent ambiguity. Qualification is defined in Appendix F, "Basic Constructs."

Identifier-3 (identifier-6) must be of the proper type to match the attribute named by attribute-name-1 (attribute-name-2). For example, if attribute-name-1 is a file attribute, then identifier-3 must be a file.

If an optional list of items to be displayed is present in Format 2, the members of the list must be separated by commas (,).

If the optional double period (..) for identifier range is used, the declaration of the first identifier must precede that of the second in the program.

General Rules

The DISPLAY command displays the values of program variables and information about declared program elements.

The output of the DISPLAY command can be discontinued by entering the CANDE ?BRK command.

Format 1

A double asterisk (**) displays all currently active program elements. Elements of program environments (for example, libraries, tasks) outside the test program are not displayed.

The data-format-option enables specification of the amount of data and the format for display. The data-format-option is defined in Appendix F, "Basic Constructs."

An attached output-command option directs the output of the display to a specified device or file.

Format 2

Specifying an expression in Format 2 evaluates the expression and displays its value.

Expressions that interrogate program variables can be specified by using the syntax defined for expressions in the *COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation.* Limitations on the types of constructs that can be used in expressions are defined under "expression" in Appendix F, "Basic Constructs."

If necessary, an identifier must include qualification to prevent ambiguity. (Qualification is defined in Appendix F, "Basic Constructs.") The name of each identifier is displayed with its usage and the sequence number of its declaration. If it is a data variable, its value is displayed.

Data within files and databases is not displayed. However, the current file record or the current user work area of the database can be displayed. Certain attributes are displayed automatically for files and tasks. For files, the automatically displayed attributes are TITLE, KIND, and OPEN. If OPEN is TRUE, then the STATE and RECORD attributes are also displayed. For tasks, including MYSELF and MYJOB, the MIXNUMBER, STATUS, and NAME attributes are displayed. Other attributes are displayed when specifically requested in Format 2.

A subscripted identifier can be specified only for a record. The number of subscripts must match the number required by the item in the record. A double period (..) can be used to represent a range within the record. An asterisk (*) represents the entire range, the subscript 1 when used before a double period or the maximum valid subscript when used after a double period.

A range of declared program elements can be displayed by using a double period (..). The range starts with the first identifier and proceeds in the order of declaration within the program symbolic through the second identifier.

Examples

Example Commands	Results
DISPLAY (X+Y), (A–B)	The expressions (X+Y) and (A–B) are evaluated and their respective values displayed.
DISP (X<Y)	The expression (X<Y) is evaluated and its value displayed.
D CONDITION-1	The Boolean value of condition-name CONDITION-1 is displayed along with the name of the variable with which it is associated.
D DISPLAYGROUP ITEMS	Each elementary item and its data (including multiple occurrences) in DISPLAYGROUP is displayed.
D DISPLAYGROUP	The data associated with DISPLAYGROUP is displayed. Elementary items and subsidiary groups are not displayed separately.
D ATTRIBUTE TITLE OF FILEX	The TITLE attribute of FILEX is displayed.
D A, B, C HEX	A, B, and C are displayed in a hexadecimal format.

continued

continued

Example Commands	Results
D PICXGROUP (4..12) EBC	Occurrences 4 through 12 of PICXGROUP are displayed as EBCDIC characters.
D COMPGROUP (*,3)	The third occurrence of COMPGROUP is displayed for each occurrence of the group to which COMPGROUP belongs.
DISP BINARY77..COMPGROUP (1,2)	Items starting with BINARY77 and ending with the record item at COMPGROUP (1,2) are displayed.
DISPLAY ** OUT P	All program elements are listed on the printer.
DISP ** OUT P, FILE1, R	All program elements are listed on the printer, written in the FILE1 disk file, and displayed at the remote station.
D BINARY-GROUPS GROUPS, HEX OUT P	All groups in BINARY-GROUPS are displayed in hexadecimal format on the printer.
AT 500 DISP COMP-ITEM-NAME OF DISPLAY-RECORD-NAME	Each time the statement at sequence number 500 is to be executed, the variable COMP-ITEM-NAME OF DISPLAY-RECORD-NAME is displayed.
DISP "A literal string"	The literal string "A literal string" is displayed.
*DISP KANJI-ITEM-NAME	The Kanji variable KANJI-ITEM-NAME is displayed.
*DISPLAY NC "<KANJI CHARACTERS>"	The Kanji literal "<KANJI CHARACTERS>" is displayed.
*D KANJI-ITEM-NAME=NC <KANJI CHARACTERS>	The expression KANJI-ITEM-NAME=NC <KANJI CHARACTERS> is evaluated and its value is displayed.
	* Kanji items or literals can be correctly displayed only on Kanji-compatible terminals or printers. For additional information, refer to the *COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation.*

DO

The DO command executes a sequence of TADS commands from a named file.

Format

```
DO file-title
```

Syntax Rules

The file-title must be the valid file title of a disk file.

General Rules

The DO command gets and executes a sequence of TADS commands from a file other than the remote file. The named file is used as a sequential source of commands.

Commands from the file are echoed at the remote station as they are executed. If an error is encountered while executing commands from the file, a message is displayed at the remote station. The user must then specify whether the remaining queued commands in the file are to be executed or discarded.

The end of a disk file record implies a blank character, but it does not mark the end of a command. Commands can cross record boundaries.

Note: *Semicolons (;) must be used to terminate all commands in a disk file. An end of file (EOF) implies a semicolon, which marks the end of a command.*

A disk file can contain DO commands, but no more than 25 disk files can be in use by DO commands at one time.

TADS uses the following file attributes when accessing the file:

- INTMODE=EBCDIC
- KIND=DISK
- DEPENDENTSPECS=TRUE.

Examples

Example Commands	Results
DO (TEST)LOAD/CONDITIONS	Commands from the disk file (TEST)LOAD/CONDITIONS are executed.
DO INIT/TEST/VARS ON TPACK	Commands from the file INIT/TEST/VARS on TPACK are executed.

ENABLE

The ENABLE command causes disabled conditions to be re-established.

Format

```
ENABLE condition-specification [ output-command ]
```

Syntax Rules

The minimum abbreviation for ENABLE is ENAB.

The ENABLE command re-establishes the conditions explicitly specified, or all conditions that have not been discarded if an asterisk (*) is specified.

An ordinal refers to the system-generated identification number for a condition. A condition label refers to the optional label of a condition. Either can be used to enable a specific discarded condition.

General Rules

The results of the ENABLE command are displayed in a message. An error message is given if the specified condition is not available or if no established or disabled conditions exist.

An attached output-command option directs the output of the generated message to a specific device or file.

Examples

Example Commands	Results
ENAB *	All conditions are enabled.
ENABLE 7	Condition number 7 is enabled.
ENAB 3, XYZ, "END OF PROC"	Condition number 3, the condition labeled XYZ, and the condition labeled "END OF PROC" are enabled.

END

The END command terminates the test session and text program execution. BYE is a synonym for END.

Format

```
{ END }
{ BYE }
```

EXIT

The EXIT command causes a premature exit from a performed paragraph or section.

Format

```
EXIT
```

General Rules

The EXIT command is valid only during the execution of a section or paragraph that was invoked by the PERFORM command. The EXIT command causes the performed section or paragraph to be prematurely exited. The point of execution of the test program returns to the position from which the PERFORM command was done.

If a fault condition stops the execution of a section or paragraph invoked with a PERFORM command, the EXIT command can be used to continue the test session from the point where the PERFORM command was done.

FREQUENCY

The FREQUENCY command lists the number of times a specified statement has been executed.

Format

```
FREQUENCY [ statement-set ] [ output-command ]
```

Syntax Rules

The minimum abbreviation of FREQUENCY is FREQ.

The FREQUENCY option of the TADS compiler control option must be specified for the FREQUENCY command to be accepted.

General Rules

The FREQUENCY command lists the statement-identifier of each statement in the statement-set with the corresponding number of times that the statement has been executed. Statement-identifiers are reported either individually or as statement-identifier-ranges.

If no statement-set is specified, the entire program is considered to be the range. Definitions of statement-set, statement-identifier, and statement-identifier-range appear in Appendix F, "Basic Constructs."

An attached output-command option directs the output of the frequency statistics to a specified device or file.

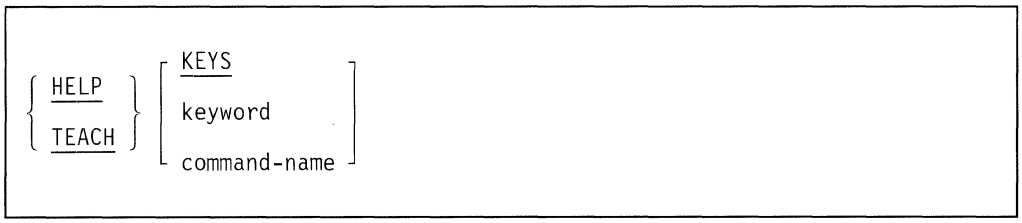
Examples

Example Commands	Results
FREQ 500.2	Frequency count for the second statement at sequence number 500 is displayed.
FREQ 900 - 1700	Frequency counts for the statement range 900 to 1700 are displayed.
FREQ OUT PRINTER	Frequency counts for the entire program are sent to the printer.
FREQ P1, P2 OUT P	Frequency counts for paragraphs P1 and P2 are sent to the printer.
FREQ S1, P3 OF S2 OUT P	Frequency counts for section S1 and paragraph P3 of section S2 are sent to the printer.
FREQ 600 @ 100 - 600 @ 200 OUT FILE1	Frequency counts for the statement-set 100 through 200 in the copy library at sequence number 600 are displayed in the FILE1 disk file.

HELP

The HELP command displays descriptions of specified TADS commands.

Format



Syntax Rules

The minimum abbreviation of HELP is H. HELP is a synonym for TEACH.

General Rules

The HELP command produces a list of all TADS commands with brief descriptions of their functions.

Specifying the KEYS keyword displays a list of all the keywords of TADS commands, and certain command-metatokens and special topics.

Specifying a keyword displays a description of the keyword. If the keyword specified is a TADS command keyword, the syntax and semantics of that command are described. Minimum abbreviations of command keywords can also be specified. A list of TADS commands and minimum abbreviations for each command can be found in Appendix A, "Command Summary."

Specifying a command-name from any of the command formats displays the description of that name. Command-name is defined in Appendix F, "Basic Constructs."

Examples

Example Commands	Results
HELP	A list of all commands with brief descriptions of each is displayed.
HELP KEYS	A list of all the keywords that can be specified for descriptions with the HELP or TEACH command is displayed.
HELP HELP	A description of the HELP command is displayed.
H C	A description of the CONTINUE command is displayed.
H expression	A definition of the term expression is displayed.

HISTORY

The HISTORY command displays the current activation history of the test program.

Format

```
HISTORY [ output-command ]
```

Syntax Rules

The minimum abbreviation of HISTORY is HIS.

General Rules

The HISTORY command displays the current activation history of the test program.

Code addresses are displayed for the following:

- The current position of execution
- The call of each external active procedure beginning with the most recently activated procedure
- Each PERFORM statement, beginning with the one performed most recently

Statement-identifiers are displayed for statements in the test program. For environments outside the test program, the name of the object file is displayed; line numbers are displayed if line information is available. If a running procedure in invoked through the PERFORM command, only the activation history up to the PERFORM command is given.

An attached output-command option directs the output of the activation history to a specified device or file.

Examples

Example Commands	Results
HISTORY	The current activation history is displayed.
HIST OUT A/B	The current activation history is written to the A/B file.
AT 400 (HISTORY; BREAK)	Every time statement 400 is reached, the activation history is displayed and execution breaks.

MERGESTAT

The MERGESTAT command merges saved and current statistics.

Format

```
MERGESTAT [ FROM ] file-title-1 [ ,file-title-2 ]...
```

Syntax Rules

The minimum abbreviation of MERGESTAT is MERGES.

The FREQUENCY option of the TADS compiler control option must be specified for the MERGESTAT command to be accepted. For additional information, see Section 2, "Preparing for a Test Session."

General Rules

The MERGESTAT command causes the statistics in the saved file or files (produced by the SAVESTAT command) to be added to the current frequency and coverage statistics. Using the COVERAGE and/or FREQUENCY commands then produces combined statistical, formatted output.

The header information in the disk file or files is checked against the current user's program for exactness before merging proceeds. If the header information does not match the current program, then the statistics in the disk file or files are not merged with the current statistics, and an error message is displayed.

Examples

Example Commands	Results
MERGESTAT FROM COV/TEST/1	The previously saved SAVESTAT file called COV/TEST/1 is added to the current statistics table for this run.
MERGES (US)COV/TEST/A ON P	This command adds statistics from the file (US)COV/TEST/A ON P to the current statistics.
MERGESTAT A/B ON C	This command adds statistics from the file A/B ON C to the current statistics.

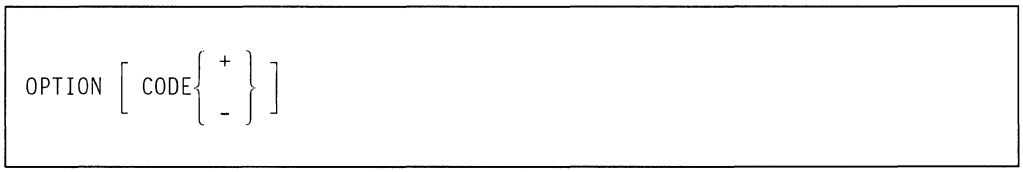
ON

The ON command, a conditional-command, specifies an expression that is evaluated before each TADS statement. When the value of the expression changes, TADS commands are executed.

OPTION

The OPTION command interrogates or changes the value of TADS options.

Format



Syntax Rules

If no option is specified, the value of each option is listed. If an option is specified, you can set the option with a plus sign (+), reset it with a minus sign (–), or set it by default if neither the plus nor the minus sign is specified.

General Rules

In addition to line information, the code address is displayed at a breakpoint and in the TRACE command. The default initial value of the CODE option is RESET.

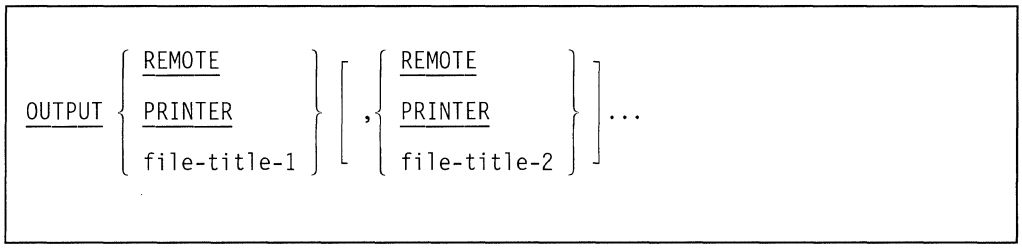
Examples

Example Commands	Results
OPTION CODE	The CODE option is SET for TRACE and breakpoint messages.
OPTION	CODE + is displayed.
TRACE	The trace information includes the code address: `Trace at 500 (BLOCK#1.BEGIN) [0067:0009:0]`
STEP	The breakpoint information includes the code address: `Breakpoint at 505 (BLOCK#1) [0067:000A:1] due to STEP 1.`
OPTION CODE -	The CODE option is RESET.

OUTPUT

The OUTPUT command directs output of TADS commands to a specified device or file.

Format



Syntax Rules

The minimum abbreviation for OUTPUT is O. The minimum abbreviation for REMOTE is R. The minimum abbreviation for PRINTER is P.

Only one specification each is allowed for REMOTE and PRINTER. If the optional list syntax is used, items in the list must be separated by commas (,).

General Rules

The OUTPUT command can be used by itself or attached to another command. When used by itself, the command changes the default routing of information for the session. When attached to another command, the OUTPUT command directs only the output of that command. Session information from any command that does not include its own OUTPUT command is routed to the default device or devices for the session.

Specifying REMOTE directs output to a remote station. REMOTE is the initial default.

Specifying PRINTER directs output to the task file of the program being tested. The task file is the same file used for program dumps.

Specifying a file title directs output to disk. No more than 25 different disk files can be specified for output during one test session. No valid abbreviation of PRINTER or REMOTE can be used as a file title. The first time output is directed to a particular disk file, the file is opened. The file remains open for successive outputs during the test session and is closed when the session terminates. When the file is opened, any other disk file with the same title is removed. Files are created with the following listed attributes. These attributes are used by Command and Edit (CANDE) to create a file of type DATA. Text occupies columns 1 through 72 of the records.

```
KIND = DISK
    FILEKIND = DATA
    UNITS = WORDS
    MAXRECSIZE = 14
    MINRECSIZE = 0
    BLOCKSIZE = 420
    AREASIZE = 1080
    SECURITYUSE = IO
    SECURITYTYPE = PRIVATE
```

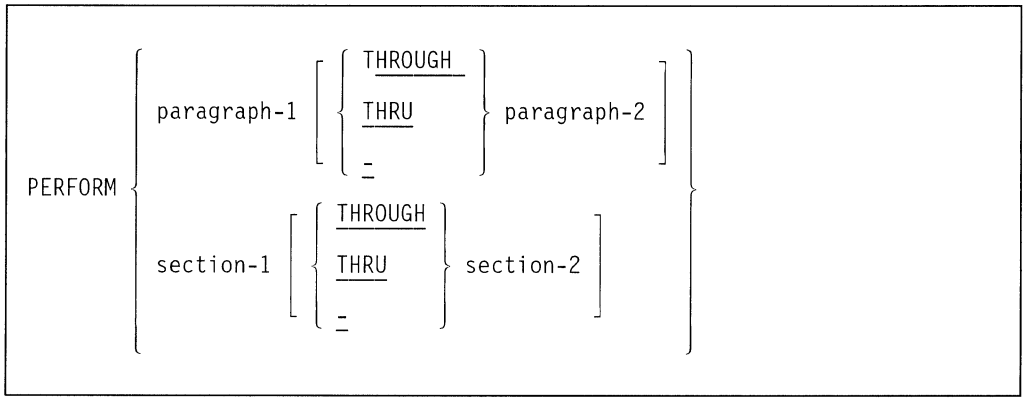
Examples

Example Commands	Results
OUTPUT PRINTER	Output is directed to the printer and the printer is assigned as the default device for this TADS session. One or more new defaults can be assigned at any time.
TRACE OUT A/B ON PK3	Trace information is directed to the A/B disk file on PK3.
TRACE	Trace information is directed to the default device or devices.
OUT R, P	Output is directed to both the remote station and the printer. The remote station and the printer are now default output devices.

PERFORM

The PERFORM command invokes a paragraph or section, a range of paragraphs, or a range of sections.

Format



Syntax Rules

The minimum abbreviation of PERFORM is P. A hyphen (-) can be used in place of THROUGH.

If the THROUGH (THRU, or -) option is present, the first paragraph or section in the range must precede the last paragraph or section in the range.

General Rules

Any section, section range, paragraph, or paragraph range can be performed, except for SORT or external procedures. This includes paragraphs and sections in the DECLARATIVES section. The PERFORM command can be used at a breakpoint but not as part of a conditional-command.

Upon completion of a paragraph, section, or a range of paragraphs or sections invoked by the PERFORM command, control is returned to the remote station. The test program returns to the point of execution at which the PERFORM command was entered.

Any TADS conditions that are met during the PERFORM operation behave as if they occurred during normal program execution. Control can be returned to the program under test by using the EXIT command.

Examples

Example Commands	Results
PERFORM P1	Paragraph P1 is performed.
PERFORM P1 - P4	Paragraphs P1 through P4 are performed.
P S1	Section S1 is performed.
PERFORM S1 THRU S2	Sections S1 through S2 are performed.
P P1 THROUGH P10	Paragraphs P1 through P10 are performed.

SAVESTAT

The SAVESTAT command saves the current statistics information.

Format

```
SAVESTAT [ AS ] file-title
```

Syntax Rules

The minimum abbreviation of SAVESTAT is SAVES.

The FREQUENCY option of the TADS compiler control option must be specified for the SAVESTAT command to be accepted. For additional information, see Section 2, "Preparing for a Test Session."

General Rules

The SAVESTAT command causes the current accumulated frequency and coverage statistics for the entire user program to be written to the specified disk file for later use by the MERGESTAT command.

In addition to containing the statistics table, the disk file also contains header information, such as the code file title and the creation date and time of the user's program. The header information is used to identify an allowable file when a file is merged using the MERGESTAT command.

Files created by the SAVESTAT command cannot be displayed by using the Editor or the CANDE *LIST* command. Such files can be displayed only by using the COVERAGE and FREQUENCY commands in a TADS session. Output from the FREQUENCY and COVERAGE commands can be routed to any combination of remote, disk, or printer backup files. For additional information, refer to the OUTPUT command.

The MERGESTAT, SAVESTAT, and COVERAGE commands can be used to accumulate frequency and coverage statistics for a series of program runs. For example, these commands might be used in the following sequence:

1. Break at the end of the user program. Use the SAVESTAT command to save program frequency and coverage statistics.
2. End the current TADS session.
3. Start a new TADS session on the same user program. Use the MERGESTAT command to read in the program frequency and cover age statistics stored in the file created by the SAVESTAT command in the previous TADS session.
4. Break at the end of the user program. Use the COVERAGE or FREQUENCY command to get statistics for the program. The statistics contain results for what is essentially two program runs combined into one.

Examples

Example Commands	Results
SAVESTAT AS COV/TEST/1	This command causes the current statistics to be copied to the file COV/TEST/1.
SAVES C/T ON DISK	This command causes statistics to be copied to the file C/T ON DISK.
SAVESTAT (US)A/B ON P	This command causes the statistics table to be copied to the file (US)A/B ON P.

SET

The SET command assigns values to program variables or to attributes, such as file, task, and database attributes.

Format 1

```
SET { identifier-1 [ .. identifier-2 ]                          }
    { ATTRIBUTE attribute-name-1 OF identifier-3                }

[ { identifier-4 [ .. identifier-5 ]                            } ]... TO { identifier-7 }
[ { ATTRIBUTE attribute-name-2 OF identifier-6                  } ]       { expression   }
```

Format 2

```
SET ATTRIBUTE attribute-name-1 OF identifier-3
[ ,ATTRIBUTE attribute-name-2 OF identifier-6 ]...TO VALUE
attribute-value-mnemonic
```

Syntax Rules

Identifiers can be subscripted and/or qualified.

The number of subscripts must match the number required by the item in the record. A double period (..) can be used to represent a range of subscripts. An asterisk (*) represents the entire range, the subscript 1 when used before a double period or the maximum valid subscript when used after a double period.

If necessary, qualification must be included to prevent ambiguity. Qualification is defined in Appendix F, "Basic Constructs."

If an optional list of items to be set is present in Format 1, the members of the list must be separated by commas (,).

If the optional double period (..) for identifier range is used, the declaration of the first identifier must precede that of the second in the program.

The types of the source and destination or destinations must match according to the tables in Appendix B, "COBOL74 TADS Compare and Set Rules." If they do not, an error is given and no items are set. For matching purposes, the type of the first item in the destination list is used.

If Format 2 is used, the attribute-value-mnemonic must match the attribute-name-1. For example,

- *SET ATTRIBUTE KIND OF FYLE TO VALUE PRINTER* is correct.
- *SET ATTRIBUTE MYUSE OF FYLE TO VALUE PRINTER* is an error.

For more information about attributes and mnemonics, see the *File Attributes Programming Reference Manual*, and the *Task Attributes Programming Reference Manual.*

Some types of items cannot be set. The types of variables that can be changed are listed in Appendix D, "Supported Data Types."

General Rules

The SET command changes the values of program variables or the values of attributes.

Identifiers can be subscripted and/or qualified. A range of identifiers and/or subscripts can be specified. Only elementary identifiers of type BINARY or REAL can be referenced by partial word syntax.

A range of program elements can be specified as identifier-1..identifier-2, using a double period (..). The range starts with the first identifier and proceeds in the order of declaration within the program symbolic through the second identifier.

Partial word syntax is valid only for BINARY items of length 11 or less and REAL items. A partial word is specified as [integer-expression-1:integer-expression-2]. The portion integer-expression-1 represents the starting bit in the word and must be in the range of 0 to 47, inclusive. The portion integer-expression-2 represents the number of bits and must be in the range of 1 to 48, inclusive (for example, the TADS command SET ID-1[12:8] TO ID-2[37:8]).

A SET command that includes a list of receiving items causes all the listed receiving items to be assigned the specified sending item or value.

Examples

Example Commands	Results
SET A, B, C, D, E to F	All the listed variables are assigned a value of F.
SET DISPREC (*) to 7	All the elements of the DISPREC record are assigned a value of 7.
SET INT1 TO INT1 + 7	Variable INT1 is assigned the value of the expression INT1 + 7.
SET BINARYREC (7..15) TO 1	Occurrences 7 through 15 of record item BINARYREC are assigned a value of 1.
SET A.. E TO 25	Program elements from A through E (in the order declared) are assigned a value of 25.
SET ATTRIBUTE TITLE OF IN TO "A/B ON C"	The TITLE attribute of the IN file is assigned the value "A/B ON C".
SET B1[3:4] TO 5	The four low-order bits of B1 are assigned a value of 5.
SET A(*,3), B(*,3) TO 0	The third item in each occurrence of record items A and B is assigned a value of 0.
SET COMPREC TO @FFFF@	COMPREC is set to four hexadecimal Fs plus enough hexadecimal 0s to fill it.
SET ATTRIBUTE KIND OF OUTFILE TO DISK	The KIND attribute of OUTFILE is assigned the value of the DISK attribute-value mnemonic.
SET DISPREC (1,2..50) TO "ABC"	In the first occurrence of DISPREC, items 2 through 50 are assigned the value "ABC".

STEP

The STEP command continues test program execution for a specified number of statements before breaking.

Format

```
STEP [ ordinal ] [ OVER ]
```

Syntax Rules

The minimum abbreviation of STEP is S.

The command can be executed only from a breakpoint; it has no meaning as part of a conditional-command.

General Rules

The STEP command continues test program execution for a specified number of statements. A breakpoint is caused after the number of test program statements specified by the ordinal have executed; 1 is assumed if the ordinal is not specified.

Statements are counted dynamically. Each statement that begins execution is counted. For example, from a breakpoint immediately preceding a PERFORM statement, the command STEP 1 causes a new breakpoint at the first statement in the performed paragraph.

If OVER is specified, then only statements executing in the paragraph or section where the STEP command is established are counted. If the end of the paragraph or section is encountered and the number of statements has not been reached, the STEP command is discarded and the number of steps taken is displayed. From a breakpoint at the end of the paragraph or section (that is, in TADS syntax paragraph-name.END), the command STEP 1 causes a new breakpoint after the implied END statement of the paragraph or section (that is, before the next statement). When the paragraph is the last in a section, a STEP 2 command is required because an END statement is also implied for the section.

Statements are counted only for the task that is suspended at the breakpoint from which the command is given. Statements in code external to the test program are not counted.

If, after the STEP command is used, a breakpoint occurs for the same task before the specified number of statements has executed, the STEP command is discarded.

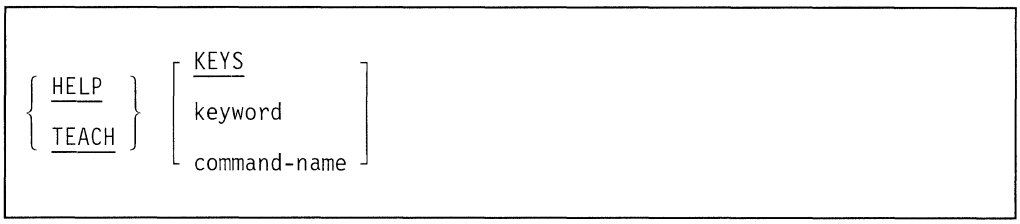
Examples

Example Commands	Results
STEP	Execution continues and then breaks after one statement.
S 7	Execution continues and then breaks after seven statements.
STEP 5 OVER	Execution continues and then breaks after five statements in the current paragraph or section, or at the end of the paragraph or section, whichever comes first.

TEACH

The TEACH command displays descriptions of specified TADS commands.

Format



Syntax Rules

TEACH is a synonym for HELP.

General Rules

The TEACH command produces a list of all TADS commands with brief descriptions of their functions.

Examples

Example Commands	Results
TEACH	A list of all the commands with brief descriptions of each is displayed.
TEACH KEYS	A list of all the keywords that can be specified for descriptions with the TEACH or HELP commands is displayed.
TEACH TEACH	A description of the TEACH command is displayed.
TEACH C	A description of the CONTINUE command is displayed.
TEACH expression	A definition of the term expression is displayed.

TRACE

The TRACE command identifies the current position of program execution.

Format

```
TRACE [ output-command ]
```

Syntax Rules

The minimum abbreviation for TRACE is T.

General Rules

The TRACE command displays the statement-identifier associated with the current point of execution in the test program. If the TRACE command is part of a conditional-command, the system-generated identification number of the condition is displayed. If the condition has a label, the label is also displayed. The MIXNUMBER and JOBNUMBER attributes of the current task are displayed if they are different from those at the initial breakpoint.

An attached OUTPUT command directs the output of the trace information to a specified device or file.

Examples

Example Commands	Results
WHEN TRUE TRACE	A complete trace of execution flow is displayed.
TRACE	The statement-identifier of the current point of execution is displayed.
AT 9000 WHEN J = 3 TRACE	At each execution of statement 9000, if the value of program variable J is equal to 3, then 9000 and the system-generated identification number for the condition are displayed.
AT P1 OF S3 LABEL L1 T OUT P	At each statement in paragraph P1 of S3, the statement identifier and label L1 are sent to the printer.
TRACE AT S2.BEGIN	Each time section S2 is entered, the statement-identifier and the system-generated condition are displayed.

WHEN

The WHEN command, a conditional-command, specifies when TADS commands are to be executed.

Section 6
Example TADS Session

This section presents a TADS session run from a Command and Edit (CANDE) window that demonstrates the use of TADS to test and debug a simple COBOL74 program. The TADS commands and the computer responses are explained.

User input to TADS and CANDE is highlighted in bold print, with the exception of the source program itself. The TADS or system responses follow immediately. A response from TADS might be another question mark (?) prompt, a result of a TADS command, or an error message. A response from the system is prefaced by #.

```
#A16:503 CANDE 35.240 AT OURSYSTEM
  #SESSION 0296  09:03:21  04/08/85

  MAKE TADS/LOAN/AMORTIZATION C74
  #WORKFILE TADS/LOAN/AMORTIZATION: COBOL74

  100$SET TADS (FREQUENCY, REMOTE LOAN-REPORT)
  200 IDENTIFICATION DIVISION.
  300    PROGRAM-ID.  TADS-LOAN-AMORTIZATION.
  400 ENVIRONMENT DIVISION.
  500 I-O SECTION.
  600 FILE-CONTROL.
  700    SELECT LOAN-REPORT  ASSIGN TO REMOTE.
  800 DATA DIVISION.
  900 FILE SECTION.
  1000 FD LOAN-REPORT.
  1100 01 LOAN-RECORD PIC X(72).
  1110
  1200 WORKING-STORAGE SECTION.
  1300 01 MONTHLY-INTEREST-RATE PIC 9V99 VALUE 0.01.
  1400 01 PAYMENTS-PER-YEAR PIC 99 VALUE 12.
  1500 01 SIMPLE PIC 9V99.
  1600 01 LOAN-PAYMENT-RECORD COMP.
  1610    03 BALANCE PIC 9999V99.
  1620    88 PAID-OFF VALUE 0.
  1700    03 TOTAL-PAYMENTS PIC 9999.
  1800    03 PRINCIPLE-PAYMENT PIC 99V99.
  1900    03 INTEREST-ON-BALANCE PIC   99V99.
  2000    03 LOAN-PAYMENT PIC   99V99.
  2010
  2020 PROCEDURE DIVISION.
```

```
3000
3010 BANK SECTION.
3020 LOAN-OFFICER.
3030    PERFORM INITIALIZE-LOAN.
3040    PERFORM PAYMENT-SCHEDULE.
3050    PERFORM END-OF-LOAN.
3060
3100 INITIALIZE SECTION.
3200 INITIALIZE-LOAN.
3300    MOVE 0 TO TOTAL-PAYMENTS.
3400    MOVE 0 TO PRINCIPLE-PAYMENT.
3500    MOVE 0 TO INTEREST-ON-BALANCE.
3510    MOVE 1000 TO BALANCE.
3600    COMPUTE SIMPLE = ((1 + MONTHLY-INTEREST-RATE) **
3710                        (- PAYMENTS-PER-YEAR)        ).
3800    COMPUTE LOAN-PAYMENT =
3900            BALANCE / ((1 - SIMPLE) / MONTHLY-INTEREST-RATE).
4000
4100 PAYMENTS SECTION.
4200 PAYMENT-SCHEDULE.
4300    PERFORM PAY-OFF VARYING TOTAL-PAYMENTS FROM 1 BY 1
4400       UNTIL PAID-OFF.
4500 PAY-OFF.
4600    IF (BALANCE + INTEREST-ON-BALANCE IS < LOAN-PAYMENT) THEN
4700       ADD BALANCE, INTEREST-ON-BALANCE GIVING LOAN-PAYMENT.
4800    SUBTRACT INTEREST-ON-BALANCE FROM LOAN-PAYMENT GIVING
4900       PRINCIPLE-PAYMENT.
5000    SUBTRACT PRINCIPLE-PAYMENT FROM BALANCE GIVING
5010       BALANCE.
5100 END-OF-LOAN.
5200 STOP-RUN.

SAVE; MAKE PAYMENTDISPLAY
#UPDATING
#WORKSOURCE TADS/LOAN/AMORTIZATION SAVED
```

A file of TADS commands, entitled PAYMENTDISPLAY, is created for use with the DO command during the TADS session.

```
#WORKFILE PAYMENTDISPLAY: SEQ

  100 DISPLAY LOAN-PAYMENT, PRINCIPLE-PAYMENT, INTEREST-ON-BALANCE;

  #

  SAVE
  #UPDATING
  #WORKSOURCE PAYMENTDISPLAY SAVED

  COMPILE TADS/LOAN/AMORTIZATION WITH COBOL74
  #COMPILING 0300
  #ET=21.0 PT=1.5 IO=1.1
```

```
RUN TADS/LOAN/AMORTIZATION;TADS
#RUNNING 0312
#?
Starting interactive test session for OBJECT/TADS/LOAN/AMORTIZATION
Breakpoint at 3030 (LOAN-OFFICER.BEGIN)
?
```

The preceding portion of the example TADS session lists the COBOL74 program TADS/LOAN/AMORTIZATION, shows the compilation messages and executes the COBOL74 program with the TADS task attribute equal to TRUE. A breakpoint is taken before execution of the first statement in the program. A question mark (?) prompt is displayed for entry of the first TADS command.

Note: *The question mark (?) prompt appears after the execution of a command and is a request for other TADS commands.*

```
?

  PERFORM INITIALIZE-LOAN
  #BOT 0322  PERFORMINITIALIZE
  #EOT 0322  PERFORMINITIALIZE
  Returned from PERFORM of INITIALIZE-LOAN
```

The PERFORM command initializes the loan-amortization program. The program returns to LOAN-OFFICER.BEGIN when the INITIALIZE-LOAN operation is completed.

Note that the #BOT <mix number> and #EOT <mix number> messages come from the system and indicate the beginning and ending, respectively, of the task in which TADS starts to perform the paragraph. TADS has assigned a name to the task consisting of part of the paragraph name.

```
?

  WHEN TOTAL-PAYMENTS > 4  (DO PAYMENTDISPLAY; BREAK)
```

This conditional statement specifies that whenever the expression TOTAL-PAYMENTS > 4 evaluates to TRUE, the commands in file PAYMENTDISPLAY are to be executed and followed by a program breakpoint.

Immediate high-level verification of the loan algorithm code can be done by displaying loan variable values after four loan payments. The code that actually calculates the payment schedule is exercised by performing the PAYMENT-SCHEDULE paragraph.

```
?
 PERFORM PAYMENT-SCHEDULE
 #BOT 0334  PERFORMPAYMENTSCH

 DISPLAY LOAN-PAYMENT, PRINCIPLE-PAYMENT, INTEREST-ON-BALANCE
 03 LOAN-PAYMENT                                        COMP at 2000
     PICTURE IS 99V99
     = 83.33, HEX:8333
 03 PRINCIPLE-PAYMENT                                   COMP at 1800
     PICTURE IS 99V99
     = 83.33, HEX:8333
 03 INTEREST-ON-BALANCE                                 COMP at 1900
     PICTURE IS 99V99
     = 00.00, HEX:0000
 0334 Breakpoint at 4600 (PAY-OFF.BEGIN) due to condition number 1
 ?
```

Note that the DISPLAY command from the DO file PAYMENTDISPLAY is echoed at the remote station when executed by TADS. In addition, note that the breakpoint was taken while the TADS *PERFORM* command was processing a paragraph.

After four payments have been calculated, the interest paid on the loan (INTEREST-ON-BALANCE) is still 0. It is obvious, at this point, that a serious error exists in the loan amortization code.

With the bug in the code revealed, the session can continue with the intent of finding and verifying a solution.

```
?
  EXIT
  #EOT 0334 PERFORMPAYMENTSCH
  EXITed from PERFORM of PAYMENT-SCHEDULE
```

The current PERFORM environment is exited with the TADS *EXIT* command, allowing testing to continue.

```
?
  PERFORM INITIALIZE-LOAN
  #BOT 1119  PERFORMINITIALIZE
  #EOT 1119  PERFORMINITIALIZE
  Returned from PERFORM of INITIALIZE-LOAN
  ?
  DISCARD *
  All conditions are discarded
```

To continue testing, the loan parameters are reinitialized by performing the initialization paragraph INITIALIZE-LOAN. Also, all TADS conditions controlling program execution are

discarded with the DISCARD * command. The asterisk (*) implies discard of all existing session conditions.

```
?
  AT 4600 SET INTEREST-ON-BALANCE TO BALANCE * MONTHLY-INTEREST-RATE
  ?
  AT PAY-OFF.BEGIN WHEN TOTAL-PAYMENTS = 4 (DO PAYMENTDISPLAY)
  ?
  AT PAYMENT-SCHEDULE.END (DISPLAY TOTAL-PAYMENTS, BALANCE)
```

Three new conditions are entered to test the user's hypothesis that the bug in the loan payment code is a result of omitting the calculation of the interest on the loan balance (INTEREST-ON-BALANCE) in the PAY-OFF paragraph.

The first condition corrects the calculation of INTEREST-ON-BALANCE for each loan payment calculated in the PAY-OFF paragraph. This conditional statement does a TADS *SET* of INTEREST-ON-BALANCE operation to the expression BALANCE * MONTHLY-INTEREST-RATE before the statement at sequence number 4600 is executed.

The second condition provides quick verification of the *bug fix* implemented by the first condition. This conditional statement invokes the file of TADS commands, PAYMENTDISPLAY, before the first statement in PAY-OFF is executed. This is done whenever the TOTAL-PAYMENTS = 4 evaluates to TRUE.

The third condition verifies that the loan algorithm terminates correctly. This condition displays TOTAL-PAYMENTS and BALANCE at the end of the PAYMENT-SCHEDULE paragraph.

Note: *In the preceding example, the user is assumed to have some knowledge of the expected algorithm result.*

```
?
  PERFORM PAYMENT-SCHEDULE
  #BOT 1120  PERFORMPAYMENTSCH
  DISPLAY LOAN-PAYMENT, PRINCIPLE-PAYMENT, INTEREST-ON-BALANCE
  03 LOAN-PAYMENT                                        COMP at 2000
      PICTURE IS 99V99
      = 83.33, HEX:8333
  03 PRINCIPLE-PAYMENT                                    COMP at 1800
      PICTURE IS 99V99
      = 74.81, HEX:7481
  03 INTEREST-ON-BALANCE                                  COMP at 1900
      PICTURE IS 99V99
      = 08.52, HEX:0852
```

At this point, it is clear that the bug fix for calculation of INTEREST-ON-BALANCE has worked. Note that the paragraph execution does not break on the display requested in the second condition (DO PAYMENTDISPLAY), but continues until the paragraph is completely performed.

```
03 TOTAL-PAYMENTS                                          COMP at 1700
      PICTURE IS 9999
      = 0014, HEX:0014
  03 BALANCE                                               COMP at 1610
      PICTURE IS 9999V99
      = 0000.00, HEX:000000
  #EOT 1120  PERFORMPAYMENTSCH
  Returned from PERFORM of PAYMENT-SCHEDULE
```

PAYMENT-SCHEDULE has completed with the correct number of payments and a final loan balance of 0. Note that TADS is still at the first executable statement of the program, yet preliminary testing and debugging of the program has been accomplished with the PERFORM command.

```
?
  BYE
  TADS session ending
  #ET=21.3 PT=1.0 IO=0.7
```

The BYE command terminates the TADS session and, in this case, returns the user to the CANDE session. An end-of-task (ET) message appears on the screen to indicate the end of the TADS session.

```
#
  BYE
  #END SESSION 1110  ET=51.7 PT=3.2 IO=2.0
  #USER = LOAN  CHARGE = xxxx.  10:21:52  04/08/85
```

The preceding example shows the status received from the system after successful log-off with the CANDE *BYE* command.

Appendix A
Command Summary

Each command of COBOL74 TADS is listed in Table A–1 with its minimum abbreviation and a short description of its use.

Table A–1. Summary of TADS Commands

Command	Abbreviation	Description
AT	AT	Conditions the commands on a statement-set
BREAK	B	Establishes a breakpoint
BYE	BYE	Terminates the test session; synonym for END
CLEAR	CLEAR	Clears test coverage and frequency statistics
CONDITIONS	COND	Displays established conditions
CONTINUE	C	Continues execution of the test program
COVERAGE	COV	Identifies unexecuted statements
DISABLE	DISA	Disables established conditions that can be re-established later by the ENABLE command
DISCARD	DISC	Discards established conditions
DISPLAY	D	Displays values of program elements
DO	DO	Gets a sequence of commands from a file
ENABLE	ENAB	Enables (re-establishes) a disabled condition
END	END	Terminates the test session; synonym for BYE
EXIT	EXIT	Prematurely exits a paragraph or section invoked with the PERFORM command
FREQUENCY	FREQ	Displays statement frequency statistics
HELP	H	Displays information about commands
HISTORY	HIS	Displays current activation history
MERGESTAT	MERGES	Merges saved and current statistics
ON	ON	Conditions commands on changes in an expression
OPTION	OPTION	Interrogates or changes the value of TADS options

continued

Table A–1. Summary of TADS Commands (cont.)

Command	Abbreviation	Description
OUTPUT	O	Directs output of TADS commands
PERFORM	P	Performs a paragraph, a section, a range of paragraphs, or a range of sections
SAVESTAT	SAVES	Saves current statistical information
SET	SET	Assigns values to program variables
STEP	S	Executes a specified number of statements
TEACH	TEACH	Displays information about commands
TRACE	T	Identifies the current statement
WHEN	W	Conditions the commands on a conditional-expression
?BREAKPOINT	?BP	Suspends test program execution Command and Edit (CANDE) command

Appendix B
COBOL74 TADS Compare and Set Rules

Tables B–1 and B–2 describe the proposed COBOL74 TADS compare and set (move) rules. The rules are based upon the current compiler rules. Each case has been tested to verify compiler function.

Table B–1. COBOL74 TADS Compare Rules

Comparing a...	With a...	Results in...
BINARY	BINARY, COMP, DISPLAY PIC 9, or NUMERIC LITERAL	A numeric comparison of algebraic value.
	REAL or DOUBLE, or FLOATING-POINT LITERAL	A floating-point comparison. The exact precision is not guaranteed.
	DISPLAY OTHER, HEX LITERAL, STRING LITERAL, KANJI, or KANJI-LITERAL	No comparison. This comparison is not allowed.
REAL or DOUBLE, or FLOATING-POINT LITERAL	BINARY, REAL or DOUBLE, COMP, DISPLAY PIC 9, NUMERIC LITERAL, or FLOATING-POINT LITERAL	A floating-point comparison. The exact precision is not guaranteed. (TADS enables you to compare a FLOATING-POINT LITERAL with either a NUMERIC or FLOATING-POINT LITERAL even though COBOL74 does not permit it.)
	DISPLAY OTHER, HEX LITERAL, STRING LITERAL, KANJI, or KANJI-LITERAL	No comparison. This comparison is not allowed.
COMP	BINARY, COMP, DISPLAY PIC 9, or NUMERIC LITERAL	A numeric comparison of algebraic value.
	REAL or DOUBLE, or FLOATING-POINT LITERAL	A floating-point comparison. The exact precision is not guaranteed.

continued

Table B–1. COBOL74 TADS Compare Rules (cont.)

Comparing a...	With a...	Results in...
	HEX LITERAL	A hexadecimal comparison. The comparison requires that variables or literals be of the same length. (TADS enables you to compare a COMP with a HEX LITERAL even though COBOL74 does not permit it.)
	DISPLAY OTHER, STRING LITERAL, KANJI, or KANJI-LITERAL	No comparison. This comparison is not allowed.
DISPLAY PIC 9	BINARY, COMP, DISPLAY PIC 9, or NUMERIC LITERAL	A numeric comparison of algebraic value.
	REAL or DOUBLE, or FLOATING-POINT LITERAL	A floating-point comparison. The exact precision is not guaranteed.
	DISPLAY OTHER or STRING LITERAL	An alphanumeric (EBCDIC) comparison starting from left to right with blank fill.
	HEX LITERAL	A hexadecimal comparison. The comparison requires that variables or literals be of the same length. (TADS enables you to compare a DISPLAY PIC 9 with a HEX LITERAL even though COBOL74 does not permit it.)
	KANJI, KANJI-LITERAL	No comparison. This comparison is not allowed.
DISPLAY OTHER	DISPLAY PIC 9, DISPLAY OTHER, STRING LITERAL, NUMERIC LITERAL, or HEX LITERAL	An alphanumeric (EBCDIC) comparison starting from left to right with blank fill. (A comparison with a NUMERIC LITERAL requires that the literal has no decimal point. A comparison with a HEX LITERAL requires that the literal has an even number of digits.)
	BINARY, REAL or DOUBLE, COMP, FLOATING-POINT LITERAL, KANJI, or KANJI-LITERAL	No comparison. This comparison is not allowed.

continued

Table B–1. COBOL74 TADS Compare Rules (cont.)

Comparing a...	With a...	Results in...
NUMERIC LITERAL	BINARY, COMP, DISPLAY PIC 9, or NUMERIC LITERAL	A numeric comparison of algebraic value. (TADS enables you to compare a NUMERIC LITERAL with either a NUMERIC LITERAL or FLOATING-POINT LITERAL even though COBOL74 does not permit it.)
	REAL or DOUBLE, or FLOATING-POINT LITERAL	A floating-point comparison. The exact precision is not guaranteed.
	DISPLAY OTHER	An alphanumeric (EBCDIC) comparison starting from left to right with blank fill. The literal must not have a decimal point.
	HEX LITERAL, STRING LITERAL, KANJI, or KANJI-LITERAL	No comparison. This comparison is not allowed.
HEX LITERAL	COMP, DISPLAY PIC 9, or HEX LITERAL	A hexadecimal comparison. The comparison requires that variables or literals be of the same length. (TADS enables you to compare a HEX LITERAL with a COMP, DISPLAY PIC 9, STRING LITERAL, or HEX LITERAL even though COBOL74 does not permit it.)
	DISPLAY OTHER, or STRING LITERAL	An alphanumeric (EBCDIC) comparison starting from left to right with blank fill. The comparison requires that the literal has an even number of digits.
	BINARY, REAL or DOUBLE, NUMERIC LITERAL, FLOATING-POINT LITERAL, KANJI, or KANJI-LITERAL	No comparison. This comparison is not allowed.
STRING LITERAL	DISPLAY PIC 9, DISPLAY OTHER, STRING LITERAL, or HEX LITERAL	An alphanumeric (EBCDIC) comparison starting from left to right with blank fill. A comparison with a HEX LITERAL requires that the literal has an even number of digits. (TADS enables you to compare a STRING LITERAL with a STRING LITERAL or a HEX LITERAL even though COBOL74 does not permit it.)

continued

Table B–1. COBOL74 TADS Compare Rules (cont.)

Comparing a...	With a...	Results in...
	BINARY, REAL or DOUBLE,COMP, NUMERIC LITERAL, FLOATING-POINT LITERAL, KANJI, or KANJI-LITERAL	No comparison. This comparison is not allowed.
KANJI, or KANJI-LITERAL	KANJI, or KANJI-LITERAL	An alphanumeric (EBCDIC) comparison starting from left to right with blank fill.
	BINARY, REAL or DOUBLE, COMP, DISPLAY PIC 9, DISPLAY OTHER, NUMERIC LITERAL, FLOATING-POINT LITERAL, HEX LITERAL, or STRING LITERAL	No comparison. This comparison is not allowed.

The following list describes the behavior of data types not included in Table B–1.

- Data Management System II (DMSII) and transaction processing system (TPS) items of type POPULATION, COUNT, RECORD TYPE, and FIELD behave like binary items.
- Numeric-valued attributes behave like binary items.
- Pointer- and string-valued attributes behave like string literals.
- DMSII BOOLEANS and condition-names and EVENT items have a value of TRUE or FALSE and can be compared to other conditional-expressions.
- TPS transaction record arrays (declared with the OCCURS clause) and any items belonging to transaction record arrays cannot be used in expressions.
- DISPLAY OTHER includes elementary items declared with PIC X or PIC A clauses, edited numerics, group items of type DISPLAY, communication description (CD) entries declared without the BINARY clause, transaction records, report names, Screen Design Facility (SDF) forms, and data sets.

The following are rules for comparison:

- Numeric operands are compared by algebraic value.
- Nonnumeric operands are compared by the collating sequence from left to right. If the operands are unequal in size, comparison proceeds as if the shorter operand were extended on the right by enough spaces to make the operands equal in size.
- BINARY, REAL, and DOUBLE group items and BINARY CDs are also considered nonnumeric, but are compared from left to right without the collating sequence, using

hexadecimal zero fill to extend the shorter operand. They can be compared only to other BINARY, REAL, or DOUBLE groups; BINARY, REAL, or DOUBLE elementary items; and BINARY CDs.

- For comparison of nonnumeric and numeric operands, if the nonnumeric is an elementary data item or a nonnumeric literal, the numeric operand is treated as if it were moved to an elementary alphanumeric data item of the same size as the numeric data item. If the nonnumeric operand is a group item, the numeric operand is treated as if it were moved to a group item of the same size as the numeric data item. Only integer numeric operands are allowed in comparison with nonnumeric operands. Undigit literals cannot be compared to a numeric operand.
- Index-names are compared to each other by occurrence number.
- For an index-name and a data item (not an index data item) or literal, the occurrence number that corresponds to the value of the index-name is compared to the data item or literal.
- For an index data item and an index-name or another index data item, the actual values are compared without conversion.

Table B–2. COBOL74 TADS Set Rules

Assigning a...	To a...	Results in...
BINARY, REAL or DOUBLE, COMP, NUMERIC LITERAL or DISPLAY PIC 9	BINARY, REAL or DOUBLE, COMP, or DISPLAY PIC 9	Numeric move: decimal points are aligned with zero fill or truncation on both ends as required. The sign is moved only if the destination is signed. If the source is unsigned, a plus sign (+) is generated for a signed destination.
	DISPLAY OTHER	Alphanumeric move with left justification, blank fill, or truncation on the right as required. The source is treated as a numeric value. The source must be an integer (unscaled). The set of a DISPLAY item requires that both source and destination have the same length. The sign is not moved. The result is left-justified with blank fill.
	KANJI	No assignment. This assignment is not allowed.

continued

Table B–2. COBOL74 TADS Set Rules (cont.)

Assigning a...	To a...	Results in...
DISPLAY OTHER	BINARY, REAL or DOUBLE, COMP, or DISPLAY PIC 9	Numeric move: decimal points are aligned with zero fill or truncation on both ends as required. The sign is moved only if the destination is signed. If the source is unsigned, a plus sign (+) is generated for a signed destination. The source is treated as a DISPLAY numeric of the same length as the destination. The length must be less than or equal to 23 digits.
	DISPLAY OTHER	Alphanumeric move with left justification, blank fill, or truncation on the right as required.
	KANJI	No assignment. This assignment is not allowed.
FLOATING-POINT LITERAL, or ARITHMETIC EXPONENT	BINARY, REAL or DOUBLE, COMP, or DISPLAY PIC 9	Numeric move: decimal points are aligned with zero fill or truncation on both ends as required. The sign is moved only if the destination is signed. If the source is unsigned, a plus sign (+) is generated for a signed destination.
	DISPLAY OTHER, or KANJI	No assignment. This assignment is not allowed.
HEX LITERAL	COMP, or DISPLAY OTHER	A hex character string move operation, left justified. If the destination is COMP, zero fill is done. If the destination is DISPLAY, blank fill is done. An even number of digits is required in the source only if the destination is DISPLAY. (TADS enables you to assign a HEX LITERAL to a COMP even though COBOL74 does not permit it.)
	BINARY, REAL or DOUBLE, DISPLAY PIC 9, or KANJI	No assignment. This assignment is not allowed.

continued

Table B–2. COBOL74 TADS Set Rules (cont.)

Assigning a...	To a...	Results in...
STRING LITERAL	COMP or DISPLAY PIC 9	Numeric move: decimal points are aligned with zero fill or truncation on both ends as required. The sign is moved only if the destination is signed. If the source is unsigned, a plus sign (+) is generated for a signed destination. The source is treated as a DISPLAY numeric of the same length as the destination. The length must be less than or equal to 23 digits.
	DISPLAY OTHER	Alphanumeric move with left justification, blank fill, or truncation on the right as required.
	BINARY, REAL or DOUBLE, or KANJI	No assignment. This assignment is not allowed.
KANJI or KANJI-LITERAL	KANJI	Alphanumeric move with left justification, blank fill, or truncation on the right as required.
	BINARY, REAL or DOUBLE, COMP, DISPLAY PIC 9, or DISPLAY OTHER	No assignment. This assignment is not allowed.

In addition to the set rules described in Table B–2, note the following:

- DMSII and TPS items of type BOOLEAN can be set to the value of any conditional-expression. DMSII items of type FIELD can be set to the value of any arithmetic expression. Individually named bits in DMSII FIELD items are treated as Booleans.
- Numeric-valued attributes can be set to the value of any arithmetic-expression.
- Pointer- and string-valued attributes can be set to the value of any DISPLAY identifier or EBCDIC literal.

Appendix C
Accessing DMSII and TPS Special Constructs

COBOL74 TADS does not provide syntax for some Data Management System II (DMSII) and transaction processing system (TPS) constructs for which the compiler does. In some cases, the information is available by using alternate COBOL74 TADS syntax. This appendix provides information about equivalent constructs.

DMSII DMSTATUS

DMSII exception reporting uses the variable DMSTATUS, which is a REAL identifier with fields showing information about error conditions. TADS provides syntax for the DMSTATUS word. The various fields can be examined by using the partial word syntax.

COBOL74/DMSII Name	TADS Field
DMSTRUCTURE	DMSTATUS [47:12]
DMCATEGORY	DMSTATUS [35: 8]
DMERRORTYPE	DMSTATUS [19:16]
DMERROR	DMSTATUS [0: 1]

The value of DMSTRUCTURE indicates the unique number of the structure involved in the error. Structure numbers are available on the program listing and from the TADS DISPLAY of the identifier.

The value of DMCATEGORY indicates the category of the error, and the value of DMERRORTYPE indicates the subcategory of the category of the error. These values and their meanings are listed in the *DMSII Application Program Interfaces Programming Guide*.

TPS Compile-Time Functions

TPS provides information about TPS record items that can be accessed at compile time. Some of these are available from TADS.

COBOL74/TPS Name	TADS Equivalent
TRFORMAT	DISPLAY format-id includes number
TRSUBFORMAT	DISPLAY subformat-id includes number
TROCCURS	DISPLAY record-item includes number
TRBITS, TRDIGITS, TRBYTES	DISPLAY record-item includes equivalent PICTURE clause from which these can be inferred

TPS Record Control Items

TPS returns system-defined items in every transaction record. They have mnemonics in the compiler, but not in TADS. They can be decoded by looking at the transaction record data in HEX format and finding the correct fields. For purposes of this list, the subscripts to the record for each word of information are provided in zero-relative indexing. These fields are system-defined and future changes to the TPS software might change their location.

COBOL74/TPS Name	TADS Equivalent
TRLRBIT	Record (0) [47: 1]
TRCONTROLSIZE	Record (0) [31: 8]
TRFORMATLEVEL	Record (0) [23: 8]
TRTOTALSIZE	Record (0) [15:16]
TRBASEDTS	Record (1) [47:48]
TRUPDATEDTS	Record (2) [47:48]
TRUPDATELEVEL	Record (3) [47:20]
TRSTATE	Record (3) [27: 4]
TRFILENUM	Record (3) [13:14]
TRBLOCKNUM	Record (4) [47:32]
TROFFSET	Record (4) [15:16]
TRINPUTOFFSETUPPR	Record (5) [47: 5]

continued

continued

COBOL74/TPS Name	TADS Equivalent
TRREMOTE	Record (5) [42: 1]
TRDATASIZE	Record (5) [39:16]
TRFORMAT	Record (5) [23:12]
TRSUBFORMAT	Record (5) [11:12]
TRCOUNTER	Record (6) [11:12]
TRREJECTED	Record (7) [47: 1]
TRCOMPACTED	Record (7) [46: 2]
TRRESTARTINFO	Record (7) [44: 2]
TRINPUTOFFSETLOWR	Record (7) [42:11]
TRSUBBASE	Record (7) [31:12]
TRUSERNUM	Record (7) [19:20]

Appendix D
Supported Data Types

The following lists show supported and unsupported COBOL 74 receiving items. (All the listed receiving items–both supported and unsupported–are supported for display.)

Supported COBOL74 Receiving Items

- Any BINARY item
- Any COMP item
- Any DISPLAY item
- Any INDEX item
- Any USAGE IS INDEX item
- Any USAGE IS KANJI item
- Any REAL item
- Any DOUBLE item
- DASDL items: BOOLEAN, DATASET, FIELD
- SDF items: Form
- CD items: Any INITIAL INPUT CD, any INPUT CD, any OUTPUT CD
- TPS items: Any transaction record
- REPORT items: Report sum counter, any print line of report

Unsupported COBOL74 Receiving Items

- Any EVENT item
- Any LOCK item
- Any TASK item
- Any transaction record array (declared with the OCCURS clause)
- Any item that belongs to a transaction record array

Appendix E
General Format Notation

General format notations are syntax diagrams that show the rules for creating COBOL74 code from words and symbols. These format notations visually represent the valid combinations of words and symbols that make COBOL74 statements or clauses.

To familiarize you with format notations, this appendix describes the components of the general format notation and provides examples of using format notations to create COBOL74 entries. You can refer to the examples at the back of this appendix as you read the explanation of each component of the general format notation.

All the format notations, whether simple or complex, follow the same basic rules. Once you understand them, the format notations serve as quick references to the syntax for statements and clauses.

The general format notation shows you the following:

- Required words and optional keywords
- Words you must provide
- The order in which the items must appear
- Entries that can be repeated
- Required and optional punctuation

When more than one format notation appears for a statement or clause, each format describes the valid combinations of COBOL74 code for a particular application. For example, there are three separate formats for three separate types of addition. The formats are numbered and labeled for ease of reference.

You can construct COBOL74 statements and clauses from the components shown in the following list. Some components represent words that you use in your code. Other components symbolize the rules for constructing your code. For example, you include uppercase underlined words in your code. You do not include the braces and brackets in your code. Instead, these symbols indicate alternatives and options for the words they enclose.

- Uppercase words
- Lowercase words
- Level numbers
- Braces ({})
- Brackets ([])
- Ellipses (...)

- Separators
- Special characters

You must code the words in the order in which they appear in the general format. The words can appear in a different order only if it is explicitly stated in the explanation of the format that you can do so.

Table E–1 shows each component of the format notation and explains its meaning.

Table E–1. General Format Notation Components

Components	Explanation
Underlined uppercase words	Identify required words that you must include in the statement or clause.
Not underlined uppercase keywords	Show optional keywords that you can include or omit without affecting the meaning of the statement or clause. An optional word enclosed in braces indicates a default option.
Lowercase words	Indicate words that you must supply. If you see a number appended to a lowercase word, that lowercase word represents a particular occurrence of that word.
Level numbers	Describe numbers that you must provide to show the organization of your data.
{} (Braces)	Enclose a group of phrases stacked vertically from which you must pick only one. If braces enclose a single phrase, you can repeat the phrase. If braces enclose multiple phrases, you must pick one phrase from the list of alternatives.
[] (Brackets)	Enclose a group of phrases stacked vertically that are optional. If brackets enclose a single option, you can omit that option. If brackets enclose multiple options, you can omit all the options or you can pick one option from the list of options. Phrases with phrases are allowed. Nested phrases appear enclosed in braces or brackets within another phrase enclosed in braces or brackets.
... (Ellipses)	Represent the point at which you can repeat a part of the format. Ellipses can appear only immediately to the right of a right bracket or a right brace. The part of the format that you can repeat is the part enclosed within the preceding brackets or braces.

continued

Table E–1. General Format Notation Components (cont.)

Components	Explanation
Separators	Indicate punctuation in a sentence. Commas and semicolons are optional. You can use separators between statements if you want. (You can interchange the comma, semicolon, and space characters.) Periods are required. You must end with a period the paragraphs within the IDENTIFICATION and PROCEDURE DIVISIONS and entries within the ENVIRONMENT and DATA DIVISIONS.
Special characters	Indicate arithmetic operations or relation conditions. Special characters are not underlined in the general format notation. However, if a special character appears, it is required. The special characters include the symbols for addition, subtraction, multiplication, division, and exponentiation (+, –, *, /, **) and the symbols for less than, equal to, and greater than (<, =, >).

Example 1

The following example is the general format notation for the BLOCK clause:

```
BLOCK CONTAINS [ integer-1 TO ] integer-2 {RECORDS    }
                                          {CHARACTERS }
```

Explanation

This example illustrates the following:

- BLOCK is a word that you must enter.
- CONTAINS is an optional keyword for improving readability.
- The integer-1 TO phrase is optional. If it is used, TO is required.
- Integer-2 is a required entry. You must substitute a number for integer-2.
- CHARACTERS is the default phrase. If you do not enter either RECORDS or CHARACTERS, the compiler assumes CHARACTERS. However, you can enter CHARACTERS to improve the readability of your code. If you do not want characters and do want records, then you must enter RECORDS.

Using the preceding format notation, you can code any of the following lines. Notice that all three lines of code are equivalent to the compiler, but the full notation is easier to read.

```
BLOCK CONTAINS 400 TO 500 CHARACTERS
BLOCK CONTAINS 400 TO 500
BLOCK 400 TO 500
```

You can also code the following line. In this instance, you can choose not to use the integer-1 TO option, and you can choose RECORDS.

```
BLOCK CONTAINS 15 RECORDS
```

Example 2

The following example is the general format notation for the COMPUTE statement:

```
COMPUTE identifier-1 [ ROUNDED ] [ ,identifier-2 [ ROUNDED ] ] ...
= arithmetic-expression [ ; ON SIZE ERROR imperative statement ]
```

Explanation

This example illustrates the following:

- COMPUTE is a word that you must enter.
- Identifier-1 and arithmetic-expression are entries that you must provide.
- ROUNDED is an option. If you choose to use it, your results are arithmetically rounded.
- , identifier-2 is an option. It can be repeated with or without the ROUNDED option.
- The equals sign (=) and period (.) are required.
- The comma (,) and semicolon (;) are optional.
- ; ON SIZE ERROR imperative-statement is an option. If you use this option, the phrase ON is optional, but SIZE, ERROR, and imperative-statement are required.

Two valid statements that you could form are the following:

```
COMPUTE fruit-inventory = apples + oranges + bananas.
COMPUTE ws-fruit, print-fruit ROUNDED = fruit-inventory + order-size.
```

Appendix F
Basic Constructs

The following basic constructs appear in the formats that describe the TADS commands:

arithmetic-expression

An expression that evaluates to a numeric value. An arithmetic-expression is composed of a combination of arithmetic primary operands and valid arithmetic operators. An arithmetic operand can be a numeric elementary item, an index-name, an index-data-item, a numeric attribute, a numeric literal, a numeric figurative-constant, a numeric special register, or an arithmetic-expression. The value of any index-name in an arithmetic-expression is the integer value of the table occurrence to which it currently refers. The value of any index-data-name in an arithmetic-expression is the integer value of index-data-name in its undecoded form. See also "expression" in this appendix.

attribute

Information describing the characteristics of an entity. The TADS syntax for referring to attributes is the same as COBOL74. For example:

```
ATTRIBUTE attribute-name OF identifier
```

Some attributes are of special interest because they implement the syntax in the SELECT clause. The value of the ASSIGN clause is found in the file attribute KIND. The value of the ORGANIZATION clause is found in the file attribute FILEORGANIZATION. The value of the ACCESSMODE clause is not available from the system (it is a write-only attribute).

attribute-name

The name of any attribute of a file, task, or database. File attributes are documented in the *File Attributes Programming Reference Manual*, task attributes are documented in the *Task Attributes Programming Reference Manual*, and database attributes are documented in the *DMSII User Language Interface Programming Guide*.

attribute-value-mnemonic

The mnemonic name of an attribute value for a file, task, or database. Refer to "attribute-name," for the pertinent documents. In TADS, attribute mnemonics are not recognized in expressions.

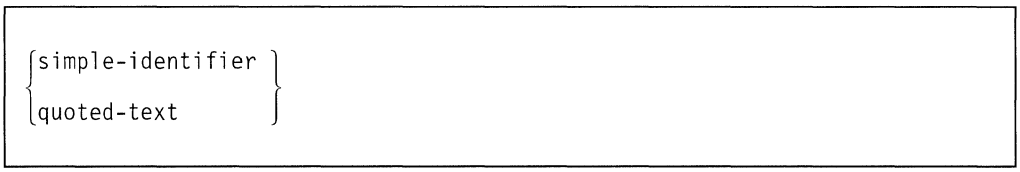
condition-accepting-command

A command that is acceptable as the action of a conditional-command. The following is a list of condition-accepting-commands. The syntax and function of each command is explained in the Section 5, "TADS Commands."

BREAK	EXIT
BYE	FREQUENCY
CLEAR	HELP
CONDITIONS	HISTORY
COVERAGE	MERGESTAT
DISCARD	OUTPUT
DISPLAY	SAVESTAT
DO	SET
END	TEACH
	TRACE

condition-label

A label used to identify a TADS condition. The condition-label format follows.



condition-name

A user-defined word assigned a specific value or range of values within the complete set of values that a conditional variable can have. A condition-name can also be a user-defined word assigned to a status of a switch or device.

conditional-command

A command used to create a TADS condition. AT and WHEN are the two conditional-commands. The syntax for conditional-commands enables the inclusion of additional conditional-commands.

conditional-expression

A condition composed of any valid combination of simple conditions and complex conditions as defined in the *COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation.* Simple conditions supported by TADS include relation-condition,

class-condition, condition-name-condition, sign-condition, and event-identifier-condition. Relation-conditions are composed of relational operands and relational operators. TADS supports the following relational operators:

- [NOT] GREATER [THAN]
- [NOT] LESS [THAN]
- [NOT] EQUAL [TO]
- [NOT] >=
- [NOT] <=
- [NOT] >
- [NOT] <
- [NOT] =

Class-conditions supported by TADS are NUMERIC and ALPHABETIC. They follow the rules in the *COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation.* The words IS and IS NOT are recognized with class and sign. Complex conditions can be formed by combining simple conditions using the relational operators NOT, AND, and OR as defined in the *COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation.*

The rules for numeric and nonnumeric comparison are included in Appendix B, "COBOL74 TADS Compare and Set Rules."

data-format-option

The option that specifies the amount of data and the format for display. The format for data-format-option follows.

```
[ GROUPS ] [ [,{HEX    }] ] ...
[ ITEMS  ]     {EBCDIC }
               {DECIMAL}
```

The minimum abbreviation for HEX is H, for EBCDIC is EBC, and for DECIMAL is DEC.

If no formatting options are specified, the minimum data for that item is displayed in a format appropriate to the type of data. If options are specified, data is displayed only in the specified formats. GROUPS displays each group in a record. ITEMS displays each group and each elementary item in a record. DECIMAL displays values as decimal numbers. EBCDIC displays data to be displayed as an EBCDIC string of characters and replaces unprintable characters by question marks. HEX displays data as a hexadecimal string. The options are ignored for program elements that do not cause data to be displayed.

expression

A combination of operands and, possibly, operators that results in the generation of a single value. An expression for COBOL74 TADS follows the syntax and semantics of

expressions in COBOL74. Expressions are evaluated using the current value of referenced variables. All operations on undefined values (error encountered in evaluation) produce undefined results with one exception: OR. Undefined operands of OR are evaluated as if they were FALSE. For example, the expression J = 1 OR K = 2 evaluates to TRUE whenever J = 1, even when K is undefined.

A COBOL74 TADS expression can be an arithmetic-expression or a conditional-expression. Unary operators are + and –. Arithmetic operators are +, –, *, /, and ** as defined in the *COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation.* Relational operators are GREATER, LESS, EQUAL, >=, >, =, <=, <, and <=. Logical operators are NOT, AND, and OR.

Precedence rules are the same as in COBOL74 and evaluate expressions in the following order:

1. Parentheses
2. Unary plus and minus
3. Exponentiation
4. Multiplication and division
5. Addition and subtraction
6. Relational operators
7. NOT
8. AND
9. OR

Arithmetic expressions are evaluated using the algorithms of the COBOL74 IF statement and might not give exactly the same results as the COBOL74 COMPUTE statement.

Attribute mnemonics are not recognized in TADS expressions. Similarly, the keyword NULL is not recognized.

figurative-constant

A certain reserved word used to name and reference a specific constant value. A figurative-constant is one of the figurative-constants defined for COBOL. ZERO (ZEROS, ZEROES) is the only numeric figurative-constant. The nonnumeric figurative-constants are SPACE (SPACES), HIGH-VALUE (HIGH-VALUES), LOW-VALUE (LOW-VALUES), and QUOTE (QUOTES). Also, any nonnumeric literal preceded by the word ALL is a nonnumeric figurative-constant. For more information, see the *COBOL ANSI-74 Reference Manual, Volume 1.*

file-title

The name assigned to identify a file. The format for file-title follows.

```
[ {*         } ]
[ {(usercode)} ]  name-1 [ /name-2 ]...[ ON name-3 ]
```

Name-1 and name-2 must be valid file directory names. Name-3 must be a valid pack name.

floating-point literal

The value of a literal shown with the decimal part of a number multiplied by the power of ten, and used as an alternative for a standard numeric literal. A floating-point literal is intended for use with REAL and DOUBLE data items, but is not restricted to that use. The format for a floating-point literal follows.

```
mantissa E exponent
```

The mantissa is the decimal part of the number. It can be signed and must have one decimal point. The exponent specifies a power of ten that is used as a multiplier. It can be signed and must be an integer.

The value of a floating-point literal is the mantissa multiplied by the power of ten expressed by the exponent. The internal storage is floating-point format, which does not guarantee precision.

hexadecimal-literal (undigit literal)

A character string bounded by at signs (@). The string of characters must consist of one or more characters chosen from the set of hexadecimal digits consisting of the digits 0 through 9 and the characters A through F. The characters A through F represent the decimal values 10 through 15, respectively. Hexadecimal-literals are also called undigit literals.

identifier

The declared name of a program element in the test program. If multiple program elements share the same name, qualification might be needed to identify which program element is being referenced. Qualification of program variables in TADS follows COBOL74 qualification rules. Qualification is accomplished by naming the valid qualifiers in the reverse order of declaration. Valid qualifiers of an item are all the groups that contain the item. The valid qualifier of a paragraph is the section that contains it. The identifier need not be fully qualified. Only enough qualification to make the identifier clear is necessary. When there is not enough qualification for an ambiguous reference, the reference cannot be resolved and an error occurs.

keyword

A reserved word that must be present when the format in which the word appears is used in a source program.

name

A string that contains 1 to 17 alphabetic characters (A through Z, a through z) and decimal digits (0 through 9).

numeric-literal

A literal composed of one or more numeric characters that can also contain either a decimal point or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

ordinal

One to 11 contiguous decimal digits (0 through 9) that represent an integer value greater than 0.

partial-word

Syntax for isolating a particular field of an entity. Partial words represent a single-precision number (fewer than 12 digits) and can only be used for items that are BINARY or REAL in USAGE. The format for partial-word follows.

```
identifier [ arithmetic-exp-1 : arithmetic-exp-2 ]
```

The brackets and the colon are all required. Arithmetic-expression-1 defines the leftmost bit location of the partial-word field being described. Arithmetic-expression-2 defines the length of the field.

Arithmetic-expression-1 must evaluate to an integer within the range 0 through 47, where 0 is the rightmost, or least significant, bit in the word. Arithmetic-expression-2 must evaluate to an integer within the range 0 through 48.

program element

Data items, files, paragraphs, sections, communication description (CD) entries, and databases that are declared. Declaration ensures that subsequent usage of the program element in a program is consistent with its declaration.

program-name

The name of the program for a TADS session. The program-name is the string present in the PROGRAM-ID section. If the program-name is not present, the default name COBOL-PROGRAM is assigned.

qualification of identifiers

Syntax for uniquely identifying an entity. Identifiers can be qualified by using the following format.

```
identifier-1 [ { OF } identifier-2 ]...[ { OF } identifier-3 ]
               { IN }                    { IN }

[ { OF } program-name ]
  { IN }
```

Identifier-1 is any identifier in the COBOL program. Identifier-2 is any group in which identifier-1 is included. Identifier-3 is the name of a special construct to which identifier-1 belongs. Identifiers that are in File Description (FD) entries can also be qualified by the name of the file. Identifier-3 can also be a database, data set, transaction base, transaction format, transaction subformat, transaction record, report name, or Screen Design Facility (SDF) form name. Any identifier can also be qualified by the name of the program.

qualification of paragraphs

Paragraphs can be qualified by the name of the section in which they are declared and by the name of the COBOL program.

```
paragraph-name [.{ BEGIN } ][ { OF } section-name ][ { OF } program-name ]
                 { END   }   { IN }                 { IN }
```

quoted-text

A string of 1 to 64 characters delimited at both ends by quotation marks (").

sequence-number

One to six contiguous decimal digits (0 through 9) that identify a line in a source program.

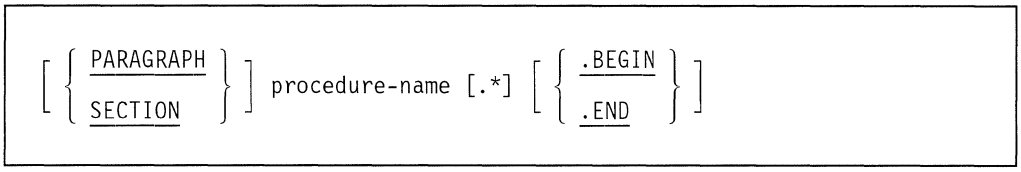
statement-identifier

Identifies a program source statement. The formats for a statement-identifier follow.

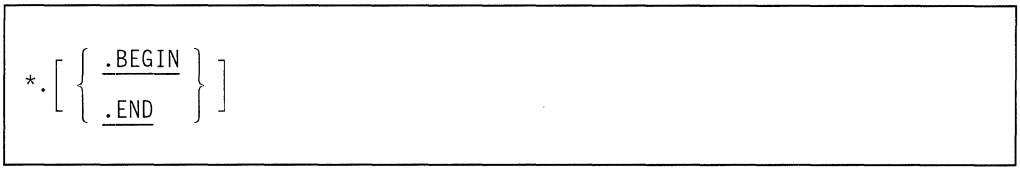
Format 1

```
sequence-number-1 [ ordinal ] [ sequence-number-2 [ ordinal ] ]
```

Format 2



Format 3



A statement identifier identifies a program source statement in one of the ways explained in the following paragraphs.

One way to identify a program source statement is to use the sequence number of the beginning of the statement. If there are multiple statements on one line, the ordinal identifies the statement by the order in which it appears. When no ordinal is specified, 1 is assumed. Multiple sequence numbers are used to represent a source line in a file included in the COBOL74 program using the COPYLIB compiler control option. Each sequence number followed by an at sign (@) represents a source record that contains a COPYLIB option, beginning with the sequence number in the program source file. The last sequence number represents the line in the included file that contains the statement being identified.

Another way of identifying a program source statement is to identify the beginning or end of a section or paragraph with the name of the section or paragraph followed by BEGIN or END. Additionally, all BEGIN or END statements in a section or paragraph are identified by the name of the section or paragraph followed by *.BEGIN or *.END. To identify all BEGIN or END statements in the program, *.BEGIN or *.END can be used without the name of the section or paragraph. This construct is especially useful for tracing program execution.

Use of *.BEGIN/END with the PARAGRAPH name is equivalent to specifying PARAGRAPH.BEGIN/END, because COBOL74 does not support nested paragraphs.

Every COBOL74 verb starts a new statement. This includes verbs that are part of ON SIZE ERROR, AT END, INVALID KEY, AT LOCKED, or ON OVERFLOW clauses. The sequence number indicates the line on which the verb appears. For this purpose, NEXT SENTENCE is considered a verb and a breakpoint can be set where NEXT SENTENCE occurs in a program.

The second way of identifying a program source statement is to identify the beginning or end of a paragraph or section with the name of the paragraph or section followed by BEGIN or END.

Since a number can be a sequence number or the name of a COBOL procedure, the optional keywords PARAGRAPH and SECTION have been added to clarify the situation. If

PARAGRAPH or SECTION precedes a numeric value in a COBOL statement, the number is treated as a paragraph or section, respectively; otherwise, it is assumed to be a sequence number. Also, if the number is followed by .BEGIN or .END (there may be spaces after the period) or the keyword OF or IN (for qualification), then the number is treated as a paragraph or section; otherwise, the number is treated as a sequence number. The PARAGRAPH or SECTION keyword is also required for a procedure-name of PARAGRAPH.

Examples	Statements Identified
3500	The first statement on line 3500
7700.2	The second statement on line 7700
700 @ 1200	The first statement on line 1200 of the copy library by way of a COPY statement at line 700 of the source program
200 @ 300.7	The seventh statement on line 300 of the copy library by way of a COPY statement at line 200 of the source program
40.5 @ 70.3	The third statement on line 70 of the copy library by way of a COPY statement at the fifth statement on line 40 of the source program
P1	Each statement in the paragraph or section P1
P1.BEGIN	The beginning of the paragraph or section P1
P3.END IN P2	The end of paragraph P3 that is declared in section P2
200.END	The end of the paragraph or section 200
400 OF P4	Paragraph 400 that is declared in section P4
SECTION 5400	Section 5400
PARAGRAPH P1	Paragraph P1 (PARAGRAPH is not needed, but allowed)
PARAGRAPH PARAGRAPH	Paragraph PARAGRAPH
SECTION 200.300	Syntax error
PARAGRAPH 2 @ 4	Syntax error
*.BEGIN	The BEGIN statements in all sections and paragraphs
SECTION P3.*.END	The end of section P3 and the end of all paragraphs in section P#

statement-identifier-range

An inclusive set of statement-identifiers defined by the beginning and ending statement-identifiers. The format of a statement-identifier-range follows.

```
statement-identifier-1 - statement-identifier-2
```

statement-set

Defines a set of statements in the test program. The set can be a single statement, a range of statements, all of the statements in a paragraph or section, or a combination.

A paragraph, section identifier, or program-name refers to statements local to the paragraph, section, or COBOL program. A paragraph or section referenced in a statement-set cannot be declared EXTERNAL.

A list of statement-sets for a simple program follows. Synonyms are grouped together.

```
100    IDENTIFICATION DIVISION.
  110    PROGRAM-ID. COBOL-PGM.
  200    ENVIRONMENT DIVISION.
  300    DATA DIVISION.
  400    WORKING-STORAGE SECTION.
  500    77 TASK-77 TASK.
  600    77 DUMMY PIC 999.
  700    PROCEDURE DIVISION.
  800    CHANGE-IT.
  900        ADD 1 TO DUMMY. ADD 2 TO DUMMY. ADD 3 TO DUMMY.
 1000        CHANGE ATTRIBUTE NAME OF TASK-77 TO "TASK77.".
 1100        CHANGE ATTRIBUTE TASKVALUE OF TASK-77 TO 77.
 1200    STOP-IT.
 1300        STOP RUN.
```

Statement Sets	Statements Referenced
900 900.1COBOL-PGM.BEGIN	The beginning of the program
900 900.1 CHANGE-IT.BEGIN	The beginning of paragraph CHANGE-IT
1000	Assignment of task name to TASK77
1100.2 CHANGE-IT.END	The end of paragraph CHANGE-IT
900.1, 900.2, 900.3, 1000, 1100, 1100.2 900 - 1100.2 CHANGE-IT CHANGE-IT.BEGIN - 1100.2 CHANGE-IT.BEGIN - CHANGE-IT.END	All the statements in CHANGE-IT, including the implied END statement.

subscript

An integer whose value identifies a particular element in a table. Identifiers that are declared with the OCCURS clause or that belong to groups with an OCCURS clause require subscripts following the same rules as COBOL74.

In addition, for the SET and DISPLAY commands, each subscript can also be an asterisk (*), which means every occurrence of that subscript of the identifier. It can also be a subscript range of the following form:

```
subscript-1.. subscript-2
```

Subscript-1 can be either an arithmetic expression or an asterisk (*). Use of an asterisk means the lowest subscript (that is, 1).

Subscript-2 can also be either an arithmetic expression or an asterisk (*). Use of an asterisk means the highest value of that subscript (that is, he number in the OCCURS clause). The double period (..) is required for a range of subscripts.

switch

Provided as a means of communicating with the external environment.

usercode

An identification code used to establish user identity, control security, and provide for segregation of files. Usercodes can be applied to every task, job, session, and file on the system.

Index

A

activation history, *See* HISTORY command
AREASIZE attribute, 5-24
attribute titles
  BLOCKSIZE, 5-24
  DEPENDENTSPECS, 5-16
  JOBNUMBER
    displayed with BREAK command, 1-2
    displayed with TRACE command, 5-33
  MINRECSIZE, 5-24
  MYSELF, 5-14
  OPEN, 5-14
  SECURITYUSE, 5-24
  STATION, 4-3
  STATUS, 5-14
  TITLE, 5-14
  UNITS, 5-24
attributes
  creating files, 5-24
  pointer-valued, B-4
  SET command usage, 5-28

B

batch mode TADS, 3-2
braces in format notation (example), E-3
brackets
  informat notation (example), E-3
breakpoints
  types, 1-2

C

characters
  undigit, F-5
comma (,)
  in format notation, E-3
commands
  from disk file, 1-5
comments
  in format notation, E-2
compare and set rules, B-1
conditional-commands
  commands excluded from, 5-5
  WHEN command, 5-5
conditional-expression
  CONDITIONAL-COMMAND, 5-4
conditions
  maintaining, 1-3
CONTINUE command
  general function, 1-2
COVERAGE command
  general function, 1-4
  with TADS compiler control option, 2-1

D

data types, D-1
data-description entry
  format of level numbers in, E-2
debug commands, 3-2
default
  in format notation, E-2
DEPENDENTSPECS attribute, 5-16
DISABLE command
  examples, 5-10
DISCARD command
  examples, 5-11
  general function, 1-4
DISPLAY command
  attributes automatically displayed, 5-14
  with subscripts, 5-14
displaying
  values, *See* DISPLAY command
DMSII and TPS special constructs, C-1
DO command
  file attributes used, 5-16
  invoking TADS commands, 3-2

E

ellipses, in format notation (example), E-3
ENABLE

command examples, 5-17
END command
general function, 1-5
ending a test session, 3-4
equal sign (=)
in format notation, E-3
EXECUTE command (CANDE), 3-1
execution
continuing, 5-9
statement statistics, 1-4
EXIT command
general function, 1-3

F

format notation, E-1
braces (example), E-3
brackets (example), E-3
comma, E-3
ellipses (example), E-3
general, E-1
letters appended to terms, E-2
level-numbers in, E-2
numbered, E-1
period, E-3
repeated items, E-2
semicolon, E-3
space, E-3
frequency analysis, 1-4
FREQUENCY option
effect on COVERAGE command, 5-9

G

general format notation, *See* format notation
<greater than sign (>)
in format notation, E-3

H

help text, 1-1
HISTORY command, 5-21

I

interrupting programs, 5-1
invoking the test system, 3-1

J

JOBNUMBER attribute
displayed with BREAK command, 1-2
displayed with TRACE command, 5-33

K

Kanji
characters, 5-15
keywords
when required, E-2

L

LABEL option, 5-5
less than sign (<)
in format notation, E-3
level-number
in format notation, E-2
libraries
testing, 4-2
literals
in format notation, E-2

M

maintaining conditions, 1-3
MARC, 4-1
MERGESTAT command
general function, 1-4
MINRECSIZE attribute, 5-24
minus sign (-)
in format notation, E-3
MYSELF attribute, 5-14

N

notation used in general formats, *See* format notation
numbered format notation, E-1
numeric-literal, F-6

O

ON command, 5-23
ONCE option, 5-6
OPEN attribute, 5-14
OPTION
  command examples, 5-23
OUTPUT command
  with CONDITIONS command, 5-8
  with HISTORY command, 5-21
  with TRACE command, 5-33

P

paragraphs
  interactive performing of, 1-3
  performing, *See* PERFORM command
PERFORM command, 5-25
period (.)
  using
    in format notation, E-3
PICTURE character string
  in format notation, E-2
plus sign (+)
  in format notation, E-3
procedures
  invoking, *See* PERFORM command
  stopping, *See* EXIT command
program
  data
    setting, 1-1
  variable, changing value of, *See* SET command
punctuation
  in format notation, E-3

R

receiving items
  not supported, D-1
REMOTE option
  with TADS compiler control option, 2-1
reserved words
  in format notation, E-2
routing diagnostic information, 1-5

S

SAVESTAT command
  with TADS compiler control option, 2-1
SECURITYUSE attribute, 5-24
semicolon
  in format notation, E-3
set and compare rules, B-1
SET command
  general function, 1-1
SET TABS ($ SET TADS), 2-1
SET TADS (FREQUENCY), 2-1
setting program data, 1-1
software testing, 2-1
space
  in format notation, E-3
special characters, in format notation, E-2
STATUS attribute, 5-14
syntax diagrams, *See* format notation
syntax, general command, 3-2

T

TADS, *See* Test and Debug System (TADS)
TADSSUPPORT library function, 2-1
TEACH command
  general function, 1-1
test
  session
    preparing for, 2-1
    terminating, 1-5
Test and Debug System (TADS)
  compiler control option, 2-1
  FREQUENCY compiler control option, 2-1
  messages in other languages, 2-2
  REMOTE compiler control option, 2-1
  session, example, 6-1
  task attribute, 3-1
TITLE attribute, 5-14
TPS and DMSII special constructs, C-1
TRACE command
  general function, 1-3
tracing program flow, 1-3

U

UNITS attribute, 5-24

W

WHEN command
  general function, 1-3
words
  lowercase, E-2
  optional, E-2
  required, E-2
  underlined, E-2
  uppercase, E-2

Special Characters

$ SET TADS (REMOTE FYLE), 2-2
* (asterisk) option
in CONDITIONS command, 5-8
?BP command, *See* ?BREAKPOINT command (CANDE)
?BREAKPOINT command (CANDE), 1-2
?SL (System Library) command (CANDE), 2-1